LossLeaP: Learning to Predict for Intent-Based Networking

Alan Collet*†, Albert Banchs*† and Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{alan.collet, albert.banchs, marco.fiore}@imdea.org

***Abstract*—Intent-Based Networking mandates that high-level human-understandable intents are automatically interpreted and implemented by network management entities. As a key part in this process, it is required that network orchestrators activate the correct automated decision model to meet the intent objective. In anticipatory networking tasks, this requirement maps to identifying and deploying a tailored prediction model that can produce a forecast aligned with the specific –and typically complex– network management goal expressed by the original intent. Current forecasting models for network demands or network management optimize generic, non-flexible, and manually designed objectives, hence do not fulfil the needs of anticipatory Intent-Based Networking. To close this gap, we propose `LossLeaP`, a novel forecasting model that can autonomously learn the relationship between the prediction and the target management objective, steering the former to minimize the latter. To this end, `LossLeaP` adopts an original deep learning architecture that advances current efforts in automated machine learning, towards a spontaneous design of loss functions for regression tasks. Extensive experiments in controlled environments and in practical application case studies prove that `LossLeaP` outperforms a wide range of benchmarks, including state-of-the-art solutions for network capacity forecasting.**

I. Introduction

Mobile networks are becoming more complex from a variety of facets, including the size and increasingly distributed nature of the infrastructure, the heterogeneity of the technologies and service requirements, or the emergence of new paradigms. As a result, network management is a more challenging task than ever, requiring rapid or anticipatory actions in response to the dynamics of a tangled environment. This is fostering zero-touch approaches that automate the management of network resources, by progressively limiting human intervention [1].

Intent-Based Networking. The ultimate vision for network management automation is *Intent-Based Networking* (IBN). In IBN, human controllers only dictate *intent policies* that define (*e.g.*, in natural language) what the network should do in terms of high-level objectives, without specifying how to achieve them [2], [3]. As illustrated in Figure 1, intents go through a *translation* stage, where they are rendered (*e.g.*, via natural language processing) into a form that is consumable by the network management entities. Those entities are then responsible for the *activation* stage, where suitable management decisions are taken to meet the target requirements. A final *assurance* stage includes monitoring of the system, verification that specifications are met, and triggering of potential adjustments. Ultimately, as intents are transparent to the underlying hardware and portable across technologies, IBN pushes humans out of the loop as much as possible, and limit their role to that of providers of conceptual operation guidelines –which the system is then expected to interpret and achieve in a fully automated manner.

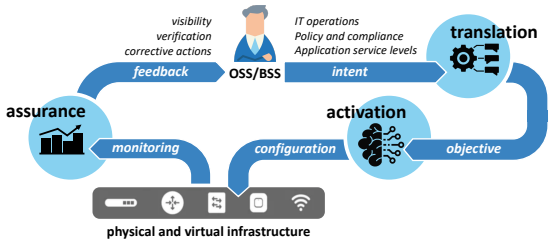


Fig. 1: High-level workflow of Intent-Based Networking.

IBN is today in its infancy, with standardization efforts that are still in progress [4]–[7]. While IBN platforms are being advertised [8], commercial solutions touting support for the paradigm are mainly network visibility tools that allow operators to retrieve monitoring information by means of intent-based declarations; they are quite far from the promises of autonomous remediation or intent-based orchestration.

Forecasting for IBN. While the attention is usually drawn to the IBN translation stage, and to its enticing prospects of controlling the network via human-understandable commands, we argue that, from a networking viewpoint, the core of the paradigm and its main technical challenges lie in the activation stage. Indeed, it is during activation that network orchestrators and controllers have to *automatically design and deploy the correct decision model to achieve the management objective*. Automatized model design is an exacting task per se. In IBN, the challenge is even harder, as objectives are machine-translated from an intent policy, hence may easily take forms that are inherently difficult to handle (*e.g.*, combining a large number of Key Performance Indicators), or do not satisfy desirable properties (*e.g.*, convexity or differentiability). These aspects set IBN activation apart from traditional human-in-the-loop policing, where the management problem is manually formalized in a way that is tractable and an appropriate decision-making model is then carefully designed to solve it.

In management contexts where decisions must be taken in advance, implementing the IBN activation stage maps to automatically tailoring and solving a suitable prediction problem for the target management objective. As a toy example, let us consider the following and deliberately simple intent policy: "ensure high reliability to all Twitch traffic streaming from the Fusion Arena in Philadelphia in the next hour". The intent would be machine-translated into a set of objectives warranting that enough transport and compute resources are proactively allocated to the Twitch network slice in the end-to-end path between the Remote Units (RUs) covering the Fusion Arena and the mobile network gateway. For instance, one such objective would target the virtualized Radio Access Network (vRAN) Far-Edge Site (ES) serving the RUs above, and require that

CPU capacity is reserved in Distributed Units (DUs) of the ES to run radio functions (*e.g.*, demodulation, decoding, forward error correction) for the entirety of the expected slice demand. Here, IBN activation consists in producing a forecast of the CPU resources needed to serve the Twitch demand in the next hour, and isolate them in the ES for exclusive use by that slice.

Assuming that a blueprint is readily available for a prediction model that solves the exact task above is not realistic: this is just one of the infinite anticipatory network management problems that may occur in practice, and building (and maintaining) a comprehensive catalog of fine-tuned models that tackle each and all of them is not possible in practical contexts. Sticking with our toy example, the original intent could change to, *e.g.*, maximizing the revenue of the operator from the Twitch slice, or allocating the minimum capacity that keeps Twitch users satisfied; these variants completely reorient the decision process, altering the notion of the *correct* amount CPU resources, and requiring a different logic to predict it.

Instead, what is ideally needed to effectively address anticipatory IBN activation problems is the ability to *automatically design a forecasting model on-demand, by aligning its operation and output to any specific management objective it is presented with*. This capacity ensures preparedness to cope with the requisites of any intent, even if not known a-priori.

Contribution. The visionary ability set out above is obviously extremely difficult to realize in practice. As we thoroughly discuss in Section II, the vast literature on forecasting models for networking only includes inflexible predictors that target a single fixed (and typically simple) objective and fail to meet the new requirement. As a step towards closing this gap, we propose `LossLeaP`, an automated approach that *learns to predict* in a way to flexibly optimize complex objectives. We discuss in Section III how the original design of `LossLeaP` allows discovering and modelling the relationship between the objective and the network system parameters, and then using it to train a dedicated predictor –hence effectively solving the problem of forecasting for IBN outlined above.

Experiments with real-world traffic in both controlled settings and practical application case studies are presented in Sections IV and V, and demonstrate the advantages and ductility of our proposed approach. As expounded in the concluding remarks in Section VI, `LossLeaP` lays the foundations to automatic tailoring of forecasting for specific complex objectives, and is a step forward towards practical IBN systems.

II. Background and Related Work

Mobile traffic prediction. Predictors for mobile network traffic have traditionally relied on statistical models, mainly based on autoregression [9]–[13]. Approaches based on tools from Markovian [14] or information [15] theory have also been proposed. More recently, deep learning has gained momentum in the design of data-driven network automation solutions [16]. Forecasting is no exception, and many recent works have employed a variety of Deep Neural Network (DNN) architectures to anticipate future mobile network loads [17]–[21], showing improved performance over previous methods.

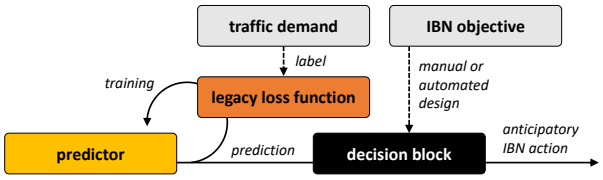


Fig. 2: Anticipatory IBN activation with traditional mobile network traffic prediction. The predictor is trained to output a pure traffic forecast that minimises the distance from the future traffic demand, measured via a legacy loss function such as MAE or MSE. A separate and subsequent decision block must then encode the actual network management objective, and post-process the prediction to produce the action.

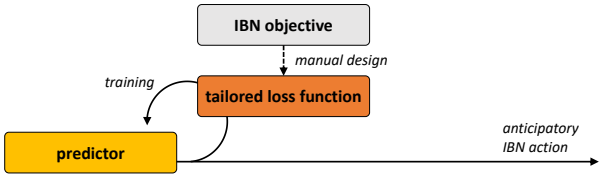


Fig. 3: Anticipatory IBN activation with capacity forecasting. The network management objective is manually encoded into a tailored loss function by human (networking and machine learning) experts. Once trained with the dedicated loss function, the predictor directly outputs the anticipatory IBN action.

All these predictors aim at producing a forecast that deviates as little as possible from the future traffic demand, by minimizing legacy error metrics such as Mean Absolute Error (MAE) or Mean Squared Error (MSE). In DNN models, as exemplified in Figure 2, this is achieved by using MAE or MSE as the *loss function*, *i.e.*, the expression that the neural network learns to minimize during training. The output provided by these predictors is completely agnostic to the (manually defined or intent-based) network management objective. Therefore, the prediction does not offer a solution to IBN activation, rather is a mere input to the actual decision-making process.

For instance, in the toy example of Section I, a traffic forecasting model approximates the future demand with some level of accuracy, incurring in both positive and negative errors. As negative errors cause underprovisioning and fail to meet the intent of reliability, the forecast cannot be used as is; instead, an overprovisioning policy must be devised *downstream of the prediction*, so as to take a CPU resource reservation decision that actually avoids underprovisioning.

Capacity forecasting. While the classical traffic predictors above factually decouple the problems of forecasting and activation, recent works on deep learning for network management have proved that jointly solving the two problems is a much more effective approach. So-called *capacity forecasting* does not predict future traffic demands, but directly anticipates the amount of resources needed to serve them [22]. As illustrated in Figure 3, a capacity forecast is achieved by training a DNN model with a loss function that encodes a specific resource management objective. In cases where meeting the IBN objective only needs allocating resources, the forecasting

model addresses the activation stage as a whole, as it directly outputs the decision needed to meet the management intent.

However, state-of-the-art models for capacity forecasting employ loss functions that are designed manually, based on expert knowledge [23]. This strategy has several limitations that make it unsuitable for IBN: (A) it requires human intervention, hence is not aligned with the vision of a full-fledged IBN activation stage where the whole decision process is automated; (B) it assumes that an effective loss function can be devised by hand, which is not the case when the relationship between the actionable network parameters and the management objective is, *e.g.*, not known a priori, or especially tangled[1]; (C) it must abide by the requisite that loss functions be differentiable, so that popular optimizers based on gradient descent can be used for training [24] – whereas machine-translated objectives may not be differentiable.

Current definitions of tailored loss functions for capacity forecasting aim at avoiding all underprovisioning by assigning a very high cost to predicting values below the demand, while penalizing additional overprovisioning [22], [23]. In fact, this suits perfectly the IBN activation toy example set out in Section I: the capacity prediction avoids underprovisioning of CPU resources, hence meets the intent objective. Yet, this is achieved via human-defined loss functions, and the approach suffers from problem (A) above. Problems (B) and (C) do not apply to our toy example, as the objective is fairly simple; however, they can easily emerge in more complicated settings.

Reinforcement learning. The general problem of automated decision-making during IBN activation naturally lends itself to be solved via Reinforcement Learning (RL). Indeed, RL does not require defining a loss function, and is the standard approach to deal with abstract but measurable objectives. However, RL is not well suited for the specific, forecasting type of task we are tackling: it operates on a limited set of well-defined discrete actions, which in our context unnecessarily restricts decisions to quantized levels. As a matter of fact, previous works where RL is employed for prediction are set in the stock market ecosystem, where the aim is anticipating price fluctuations to take simple buy or sell decisions [25], [26]. Different from stock markets, network management benefits from predictions on a continuous space, which ensures that, *e.g.*, the exact required amount of resources are allocated in concert with fluctuations in the demand. In fact, as later discussed in Section III, our proposal can be interpreted as a way to reconcile RL with the continuous-value forecasting task that is needed for IBN activation.

Loss-metric mismatch. Our work relates to the problem of loss-metric mismatch in machine learning, which stems from the observation that the loss function used to train a DNN is not always aligned with the actual metric, *i.e.*, the machine-translated management objective in our case. In a sense, the mobile traffic predictors mentioned before do not answer the loss-metric mismatch, and they produce an output that is not directly compatible with the objective. Instead, capacity forecasting models address the loss-metric mismatch in the context of network resource allocation, by using a loss function that is consistent with the actual cost of prediction errors from the operator's viewpoint.

Adaptative Loss Alignment (ALA) is a state-of-the-art machine learning approach to automatically solve the loss-metric mismatch [27]. ALA employs a linear combination of loss functions, whose weights are trained via RL so as to optimize the target metric. The resulting blend of loss functions can then be used to train a DNN model so that it responds to the precise needs of the system. However, ALA requires identifying a set of predefined loss functions that are relevant to the problem at hand. This is not an issue in the original work introducing ALA, which tackles generic classification, for which well-known loss functions are available. Yet, in network management settings, the approach hits the same hurdles (A)–(C) listed earlier for capacity forecasting models.

Automated machine learning. Recent works in mobile network management have started exploring Automated Machine Learning (AutoML) tools. AutoML aims at automatizing complex (and typically manual) tasks in the configuration of a deep learning model, such as input formatting and selection, hyperparameter optimization, or analysis of result so that they are understandable to non-experts in machine learning [28]. In particular, AutoML approaches based on Bayesian optimization using Gaussian Processes have been used to optimize the configuration of the network so as to best support particular types of services. While it indeed automates the network management process, that specific solution does not yet fulfil the requirements of forecasting for IBN: on the one hand, it does not target anticipatory configuration based on a prediction, rather optimizes the system parameters reactively; on the other hand, it imposes strong conditions on the shape of the target metric, like bounded norms and Lipschitz continuity, not easily met in generic machine-translated objectives.

Yet, the AutoML vision is well aligned with the needs of IBN activation. As explained next, our work indeed contributes to advancing AutoML methods towards automatizing the training of DNN-based forecasting models so that they fit the requisites of complex network management objectives.

III. Loss-Learning Predictor

In order to close the gap between the requirements of forecasting for IBN set out in Section I and the literature on traffic and capacity prediction presented in Section II, we propose `LossLeaP`, a Loss-Learning Predictor. The concept, architectural implementation, and training process of our original machine learning approach to forecasting are detailed next.

A. Concept

The basic idea underpinning the design of `LossLeaP` is simple: instead of imposing a predefined expression of the loss function used to train a DNN predictor, we let the forecasting model free to learn the loss function that best suits the network management objective at hand. In practice, this is realized

[1]For instance, loss functions proposed for capacity forecasting present a piecewise linear design, and cannot capture non-linear or multivariate objectives.

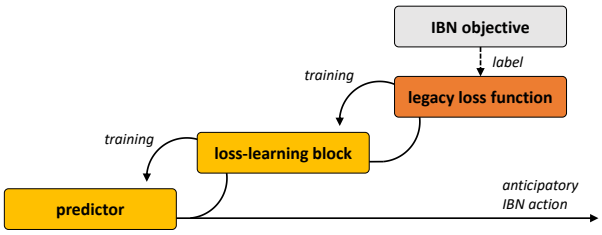


Fig. 4: Anticipatory IBN activation with LossLeaP. The network management objective is learnt and encoded into a loss-learning block. This block then serves as the loss function to train the predictor, so that it directly outputs the anticipatory IBN action. The whole process is automated.

by combining a *loss-learning block* with the actual predictor, as shown in Figure 4. This block is responsible for learning the loss function, or, equivalently, capturing the relationship between the forecast produced by the predictor and the target management objective. Once ready, the loss-learning block can operate as a tailored loss function: it receives the output of the DNN predictor and determines its quality for the precise management task. Therefore, it can be employed to train the DNN predictor so as to steer the optimization of its parameters towards minimizing the actual IBN objective.

Our choice for the implementation of the loss-learning block is a second DNN, in cascade to the first one that performs the prediction. The loss-learning DNN can be trained as a regular neural network, by minimizing the MSE between its output and the objective. It is worth noting that such a *loss training can use performance measurements collected in the target system as a direct representation of the objective*, without any need to formalize it as a mathematical function. In this sense, LossLeaP can be seen as an RL-like approach in spirit, as it learns by observing the outcome of its decisions; however, unlike RL methods, it solves a regression problem and outputs a continuous-valued action. This design allows removing at once all the limitations of previous forecasting approaches.

- The loss-learning DNN can learn the relationships between the prediction and the objective from measurement data, without need for human intervention, solving (A).
- Without any need for prior knowledge of the system, the loss-learning DNN can model tangled non-linear and multivariate objectives that may result from the translation of IBN intents, which addresses problem (B).
- The only prerequisite on the objective is that its values should be minimum to attain the best performance, which is very supple: if not implicit, the requirement can be met with naive transformations of the performance measurement data during training; this removes problem (C).

For instance, when applied to the toy example in Section I, LossLeaP would learn a loss function that is very similar to that manually defined by experts for capacity forecasting [22], [23], but it would do so in a fully automated way.

Overall, LossLeaP contributes to advancing AutoML methods towards automating the design of loss functions –a direction still largely unexplored in the machine learning domain.

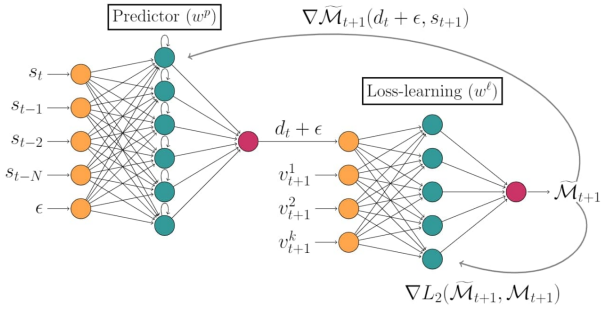


Fig. 5: Architecture of LossLeaP.

B. Detailed implementation

While the conceptual design of LossLeaP is plain enough, its effective implementation includes several gimmicks that are summarized in the detailed architecture of the model, portrayed in Figure 5. As already mentioned, the model is composed of two DNNs in cascade, which are respectively responsible for predicting the network management action, and learning the correct loss function to train such a prediction.

Co-training of predictor and loss-learning block. A first important observation is that, in the actual implementation of LossLeaP, the two DNNs are placed in series, with the output of the predictor fed directly to the loss-learning block. From a practical viewpoint, this allows optimizing the two DNNs jointly, by using a single backpropagation process; in other words, during training, the model in Figure 5 can update the loss function, and then use it to tune the predictor's weights within the same gradient descent iteration.

Formally, let w^p_t and w^ℓ_t be the weights of the predictor and loss-learning DNNs at training time t, respectively. These weights are optimized as follows. During the forward pass, N (normalized) past observations $\bar{s}_t = \{s_{t-N}, \ldots, s_t\}$ of the system state[2] are input to the predictor, which generates the anticipatory network management decision d_t to be enacted during the following time step. The performance of the decision taken at time t, denoted by $\mathcal{M}_{t+1} = f_{\mathcal{M}}(d_t, \bar{v}_{t+1})$, is measured[3] at time $t+1$; at the same time, the set of current observations $\bar{v}_{t+1} = \{v^1_{t+1}, \ldots, v^k_{t+1}\}$ of the system variables that may affect $\mathcal{M}_{t+1}$ is also collected. The forward pass can then be completed, by feeding d_t and $\bar{v}_{t+1}$ to the loss-learning block, and generating a performance estimate $\widetilde{\mathcal{M}}_{t+1}$; let $L_{w^\ell_t}$ be the (loss) function implemented by the loss-learning DNN with optimized weights at time t, hence $\widetilde{\mathcal{M}}_{t+1} = L_{w^\ell_t}(d_t + \epsilon, \bar{v}_{t+1})$. The error between the actual and estimated objectives $\mathcal{M}_{t+1}$ and $\widetilde{\mathcal{M}}_{t+1}$ is computed via a legacy loss function such as MAE or MSE, and is then backpropagated: first in the loss-learning block, updating w^ℓ_{t+1} so that they better capture how $\mathcal{M}_{t+1}$ relates to d_t and $\bar{v}_{t+1}$; then through the predictor, updating w^p_{t+1} to better model the anticipatory decision d_t that minimizes the objective $\mathcal{M}_{t+1}$.

[2]Our definition of system status is generic, and $\bar{s}_t$ can map to, *e.g.*, data traffic volumes, KPI values, or previous decisions, depending on the IBN task.

[3]Recall that, as explained in Sections I and II, in practical IBN systems the expression $f_{\mathcal{M}}$ of the objective is not known a priori, or is too complex to be modelled manually, but the associated performance can be measured.

As an illustrative example, let us consider again the toy use case of Section I. There, $\bar{s}_t$ is the traffic volume generated by the Twitch slice until time t, which is used by the predictor to generate d_t, *i.e.*, the amount of CPU resources allocated during time slot $t+1$. The loss-learning block can just observe the actual Twitch traffic volume at time $t+1$, *i.e.*, $\bar{v}_{t+1} = s_{t+1}$, and the performance $\mathcal{M}_{t+1}$. Through the training iterations, the loss-learning block will learn that $\mathcal{M}_{t+1}$ is a specific function of the difference between the allocated and needed resources at time $t+1$, *i.e.*, it has an expression $f_{\mathcal{M}}(d_t - k_{\text{CPU}} \cdot s_{t+1})$, where k_{CPU} is the CPU needed to serve one unit of data traffic; specifically $f_{\mathcal{M}}$ will be very high in case of underprovisioning, *i.e.*, if $d_t < k_{\text{CPU}} \cdot s_{t+1}$. By further descending the gradient to the predictor, the latter will progressively learn to output a minimum safe value of capacity d_t such that $d_t \geq k_{\text{CPU}} \cdot s_{t+1}$ is always verified, even in presence of typical forecasting errors.

The advantage of the co-training described above is twofold: (i) the loss-learning block can discover the actual relationship between the imperfect prediction and the IBN objective, hence it learns a loss function that implicitly compensates for forecasting errors; and, (ii) the loss-learning DNN approximates non-differentiable objectives with a differentiable version, which can then be used to train the predictor.

Noisy exploration during training. A second key trait of the architecture in Figure 5 is that the decision value input to the loss-learning block at time t is actually $d_t + \epsilon$, rather than the plain d_t output by the predictor. Here, ϵ is a normally distributed random variable that is added to the anticipatory action issued by the predictor; note that this is only adopted during training, and $\epsilon = 0$ when `LossLeaP` is used for inference. The purpose of ϵ is to explore how the system performance $\mathcal{M}$ reacts to prediction values that would otherwise never be seen during training: thanks to the added noise, the loss-learning block can learn more reliably the shape of the loss function over a wider domain.

Formally, under the noisy exploration provoked by ϵ, the gradient descent updates weights in the predictor and loss-learing DNNs as follows:

$$w^p_{t+1} \leftarrow w^p_t - \alpha^p \nabla L_{w^\ell_t}(d_t + \epsilon, \bar{v}_{t+1}), \tag{1}$$

$$w^\ell_{t+1} \leftarrow w^\ell_t - \alpha^\ell \nabla L_2(L_{w^\ell_t}(d_t + \epsilon, \bar{v}_{t+1}), f_{\mathcal{M}}(d_t + \epsilon, \bar{v}_{t+1})), \tag{2}$$

where L_2 is the MSE loss used to compare the estimated and actual performance, and α^p and α^ℓ are the learning rates of the predictor and loss-learning DNNs, respectively.

However, ϵ factually introduces an inconsistency between the output of the predictor and the input of the loss-learning block; this would confound the training of the former, which would generate decision d_t that try to compensate for ϵ. To rectify the learning process, the same value of ϵ that is summed to d_t is also provided as a further input to the predictor along $\bar{s}_t$. In this way, during training, the predictor can aptly learn the impact of the added noise to its output; and, during inference, setting $\epsilon = 0$ is sufficient to instruct the predictor to generate an unbiased decision that is not compensating for any noise.

Cyclic Learning Rate. In practical cases, the multi-variate high-degree polynomial representation of $L_{w^\ell_t}$ learned by the loss-learning DNN does not have real flat parts or saddle points. This allows using a Cyclic Learning Rate (CLR) method [29] to dynamically adapt the learning rate used to explore the gradient of $L_{w^\ell_t}$ while optimizing the weights w^p_t of the predictor DNN. By letting the learning rate oscillate between two bounds whose values are updated at each batch, CLR prevents Stochastic Gradient Descent (SGD) from getting stuck in local minima, and makes the model less sensitive to the initial learning rate value. We employ the simple triangular version of CLR, with 5 cycles across the full training phase.

Ultimately, CLR allows for faster automatic convergence, under any shape of the actual objective function $f_{\mathcal{M}}$. It is also well aligned with the AutoML principles embraced by `LossLeaP`, as it automates further the machine learning model configuration: indeed, by autonomously adapting the learning rate, CLR reduces the importance of the initial setting, and avoids that a careless choice (*e.g.*, a default learning rate value, or one casually picked by a human operator) causes the learning process to become stuck at local minima.

Architectures of the DNNs. The `LossLeaP` implementation is general, *i.e.*, can accommodate any type of DNN to realize the predictor and loss-learning block. For our experiments, we make a specific choice of DNN architectures, which, despite its simplicity, proves very effective in all controlled experiments and practical use cases we studied.

The predictor DNN is implemented as a Recurrent Neural Network (RNN) with one Long Short-Term Memory (LSTM) layer formed by 50 neurons with standard Rectifier Linear Unit (ReLU) activation functions. The rationale for the choice of an LSTM predictor is that the estimation of the future value of d_t from past state observations $\bar{s}_t$ is essentially a prediction over time, and LSTM is known to perform well in time series forecasting tasks. As thoroughly explained before, the RNN is trained using the loss values produced by the loss-learning DNN, which we implement as a simple 3-layer MultiLayer Perceptron (MLP) composed of 15 neurons in each layer, with ReLU activation function. The MSE loss is used to train the MLP. This baseline architecture can already model fairly complex non-linear loss functions, and we argue that it can meet the needs of a wide range of IBN activation tasks.

In our experiments, a small value of $\epsilon = 0.01$, equal to 1% of the maximum normalized input, proved sufficient to explore the loss function domain. The popular Adam optimizer is used to perform the SGD during training.

In all cases, we recall that the DNN realizations above can be made more complex if needed. In fact, recent AutoML methods may be also adopted to identify and configure the correct DNN architectures for prediction and loss learning, pushing `LossLeaP` closer to a full-fledged anticipatory IBN. However, these refinements are out of the scope of this paper, whose contribution and focus is the loss-learning prediction; instead, they represent interesting directions for follow-up research towards production-ready IBN systems.

IV. Comparative evaluation

In order to demonstrate the advantages of `LossLeaP`, we next carry out a comprehensive comparative evaluation, juxtaposing our proposed model by a number of benchmarks.

A. Controlled environment use cases

As here the focus is on the relative performance of the different models, we perform tests in a controlled environment that favors the interpretability of the results. Specifically, we assume a setting where the objective of IBN activation, *i.e.*, $f_{\mathcal{M}}$ in the notation of Section III, is known and simple enough to be expressed in a closed form. This allows manually designing a loss function tailored to the objective, and use it in a baseline benchmark. To test generalization, we consider four different use cases with diverse forms of $f_{\mathcal{M}}$, as follows.

Traffic forecasting under absolute error. The objective is a traditional traffic forecasting, *i.e.*, predicting a d_t that matches the future data traffic volume s_{t+1}, with an error cost that is linearly proportional to the absolute discrepancy. Formally, $\mathcal{M}_{t+1} = |d_t - s_{t+1}|$, which is optimized by a MAE loss.

Traffic forecasting under squared error. Same, but larger errors tend to have an increasingly higher cost for the operator, with a quadratic growth. Formally, $\mathcal{M}_{t+1} = (d_t - s_{t+1})^2$, which is minimized by a legacy MSE loss function.

Resource allocation with probabilistic guarantees. The IBN objective is providing a probabilistic guarantee on the anticipatory allocation of resources, so as to accommodate the future traffic demand s_{t+1} a fraction τ of the time. Formally, $\mathcal{M}_{t+1} = \tau \cdot \mathcal{R}(s_{t+1} - d_t) + (1-\tau) \cdot \mathcal{R}(d_t - s_{t+1})$, where $\mathcal{R}(x) = x \cdot \mathbb{1}_{x \geq 0}$, and $\mathbb{1}_C$ is an indicator function that takes value 1 if condition C is verified and 0 otherwise. This is a *quantile forecast* that can be optimized via a pinball loss [30].

Capacity forecasting. The goal of the operator is anticipating the capacity needed to (i) avoid an expensive monetary fee α incurred for non-serviced future traffic s_{t+1}, and (ii) limit unnecessary overdimensioning beyond s_{t+1}. Formally, $\mathcal{M}_{t+1} = \alpha \cdot \mathbb{1}_{d_t < s_{t+1}} + (d_t - s_{t+1}) \cdot \mathbb{1}_{d_t \geq s_{t+1}}$. This IBN objective maps in fact to the capacity forecasting problem addressed by previous works in the literature, for which the expert-designed α-OMC loss function is available [22].

All experiments are run on real-world traffic generated by Facebook Live, and transiting in a core network datacenter of an operational 4G mobile network; see Section V for more details on the data and its collection. We employ 9 weeks of data for training, 1 week for validation, and 1 week for testing.

B. Benchmarks

We compare `LossLeaP` with a wide range of benchmarks that include baselines, state-of-the-art models for loss learning, and variants of our proposed scheme.

1) Baseline approaches: Two different solutions are used as a basis for our comparative performance evaluation.

- **`Manual`** – The LSTM predictor described in Section III-B is trained with a loss function designed manually to fulfill the specific target objective. As anticipated, MAE, MSE, pinball and α-OMC losses are used in the four use cases.

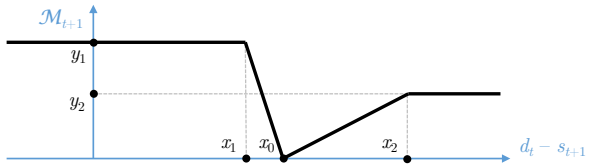


Fig. 6: Parametrizable loss function used by `ALA-moldable`.

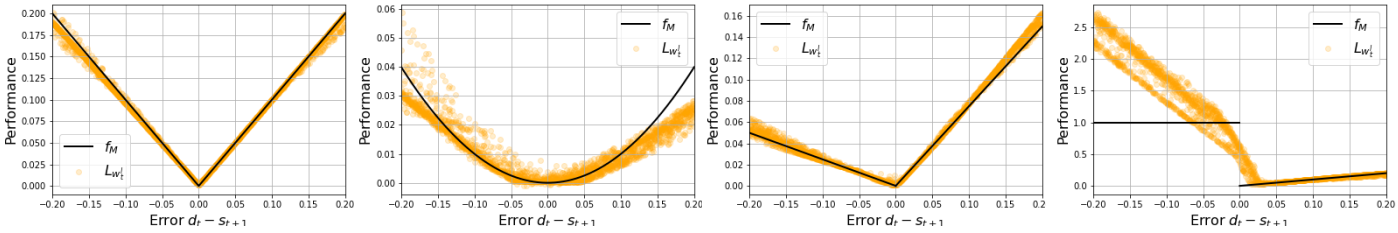
- **`Disjoint`** – The prediction and loss-learning functionalities are logically separated: first, the MLP in Section III-B is trained in isolation, by inputing uniform random noise and measuring the resulting performance, so as to learn the correct loss for the objective; then, the LSTM predictor is trained using the loss learned by the MLP.

2) Loss-learning models: As discussed in Section II, Adaptive Loss Alignment (ALA) is a state-of-the-art approach for loss learning in classification tasks [27]. We adapt ALA for regression by: (i) changing the type of characteristics used for validation, replacing the (logarithmic) probabilities that the input pertains to each class with the first- and second-order statistics of the regression values; and, (ii) swapping the set of classification-oriented loss functions originally used by ALA with expressions that are suitable for regression. We test two ALA models, which differ by the expression used.

- **`ALA-manual`** applies ALA on a linear combination of all manually designed loss functions that are used separately in the `Manual` approach above. The rationale is having ALA automatically select the correct loss function for each use case, by tuning the linear combination weights.
- **`ALA-moldable`** applies ALA on a single loss function with a highly parametrizable shape. The function, illustrated in Figure 6, can potentially mimic any of the losses for the controlled environment use cases, and the experiment aims at testing if ALA can learn the correct values of the parameters x_0, x_1, x_2, y_1 and y_2.

3) Variants of `LossLeaP`*:* As a form of ablation study, we test several variants of our proposed model, as follows.

- **`Fixed-rate`** operates as `LossLeaP`, but is trained with a legacy fixed learning rate, instead of an automated CLR.
- **`Noiseless`** uses the same architecture as `LossLeaP`, but does not include ϵ as an input to the predictor, which thus cannot learn the impact of the noise on its output; note that if no noise is added to the prediction either, the noisy exploration is completely skipped during training.
- **`Iterative`** adopts an alternating training strategy, instead of `LossLeaP` co-training. Specifically: the predictor DNN is trained in isolation during odd iterations, using the loss currently implemented by the loss-learning block; then, the loss-learning DNN is trained during even iterations, by adding noise to the output of the predictor.
- **`Half`** is a complementary technique that can be adopted in combination with the `Noiseless` and `Iterative` approaches above. In a first moment, it trains the predictor and loss-learning DNNs jointly, and as mandated by either model; then, it freezes the loss-learning block and only keeps training the predictor for better convergence.



(a) Traffic forecasting, absolute error (b) Traffic forecasting, squared error (c) Resource allocation, $\tau = 0.75$ (d) Capacity forecasting, $\alpha = 1.0$

Fig. 7: Loss functions $L_{w_t^\ell}$ learned by the loss-learning DNN of `LossLeaP`, under diverse objectives $f_{\mathcal{M}}$.

TABLE I: Comparative evaluation summary. The **best** and **second best** performing models are highlighted for each objective.

Model		Noise ϵ (deviation)	Traffic forecasting, absolute		Traffic forecasting, squared		Resource allocation, $\tau = 0.75$		Capacity forecasting, $\alpha = 1$	
			Train ($\times 10^{-2}$)	Test ($\times 10^{-2}$)	Train ($\times 10^{-4}$)	Test ($\times 10^{-4}$)	Train ($\times 10^{-3}$)	Test ($\times 10^{-3}$)	Train ($\times 10^{-2}$)	Test ($\times 10^{-2}$)
`Manual`		-	1.32 ± 0.12	1.62 ± 0.21	**2.64 ± 0.43**	4.04 ± 0.24	**4.81 ± 0.35**	**5.99 ± 0.67**	**3.30 ± 0.53**	**3.62 ± 0.61**
`Disjoint`		0.1	1.45 ± 0.11	1.80 ± 0.14	4.28 ± 0.85	6.50 ± 1.50	6.54 ± 0.42	8.26 ± 0.83	9.50 ± 1.10	14.80 ± 2.40
`ALA-manual`		-	2.28 ± 0.33	2.35 ± 0.36	6.80 ± 1.20	7.90 ± 1.70	11.20 ± 1.70	14.20 ± 2.04	8.91 ± 2.43	13.35 ± 3.17
`ALA-moldable`		-	**1.17 ± 0.04**	1.46 ± 0.04	**2.71 ± 0.22**	**3.87 ± 0.23**	5.03 ± 0.42	6.41 ± 0.55	40.04 ± 30.21	40.32 ± 30.78
`Noiseless`		0	1.71 ± 0.59	2.01 ± 0.58	5.12 ± 1.96	5.83 ± 2.29	6.13 ± 0.37	7.60 ± 1.13	6.21 ± 1.58	8.62 ± 3.55
		0.01	1.21 ± 0.13	**1.39 ± 0.14**	3.17 ± 0.90	5.30 ± 1.19	5.28 ± 0.41	6.43 ± 0.95	6.10 ± 1.27	8.88 ± 3.01
		0.1	1.19 ± 0.10	1.49 ± 0.15	3.32 ± 0.85	5.20 ± 1.20	18.04 ± 5.66	19.03 ± 6.87	17.16 ± 1.06	17.31 ± 1.30
`Iterative`	plain	0.01	1.49 ± 0.18	1.99 ± 0.21	4.10 ± 1.11	4.90 ± 1.33	6.29 ± 0.96	8.40 ± 1.90	6.67 ± 0.70	9.32 ± 1.66
		0.1	1.48 ± 0.15	2.01 ± 0.18	4.02 ± 1.02	5.82 ± 1.15	5.69 ± 0.69	7.34 ± 1.37	7.02 ± 0.63	8.12 ± 0.44
	`Half`	0.01	1.53 ± 0.11	2.01 ± 0.15	5.21 ± 1.31	15.96 ± 6.14	6.95 ± 1.98	9.09 ± 3.47	6.63 ± 1.02	7.96 ± 1.21
		0.1	1.54 ± 0.08	2.08 ± 0.16	4.85 ± 1.14	7.22 ± 2.47	6.10 ± 0.99	7.90 ± 1.65	7.65 ± 1.40	8.41 ± 1.59
`Fixed-rate`	plain	0.01	1.21 ± 0.09	1.54 ± 0.10	3.17 ± 0.75	5.80 ± 1.12	5.06 ± 0.22	6.31 ± 0.39	5.99 ± 0.80	7.94 ± 1.61
		0.1	1.31 ± 0.11	1.57 ± 0.10	4.28 ± 0.12	5.90 ± 1.01	5.25 ± 0.17	6.50 ± 0.26	6.28 ± 1.09	8.04 ± 1.33
	`Half`	0.01	1.24 ± 0.03	1.51 ± 0.02	3.72 ± 0.85	5.10 ± 0.85	5.33 ± 0.47	6.81 ± 0.61	6.05 ± 1.35	7.53 ± 1.21
		0.1	1.26 ± 0.03	1.55 ± 0.05	6.32 ± 2.18	7.79 ± 3.36	6.01 ± 0.52	6.42 ± 0.60	5.71 ± 1.04	7.94 ± 2.17
`LossLeaP`		0.01	**1.15 ± 0.01**	**1.44 ± 0.01**	2.77 ± 0.27	**4.03 ± 0.22**	**4.54 ± 0.05**	**5.63 ± 0.06**	**4.65 ± 0.13**	**5.54 ± 0.27**

C. Results

We first illustrate the capability of `LossLeaP` to learn a suitable loss function in the different use cases outlined in Section IV-A. Figure 7 portrays the four loss functions $L_{w_t^\ell}$ learned by our model from the training data, which map the error $d_t - s_{t+1}$ into the target system performance. The corresponding shape of the objective $f_{\mathcal{M}}$ is superposed to the learned loss so as to ease the interpretability of the result. In all cases, the match is good, as the two shapes are well aligned.

It is worth remarking that the only significant difference emerges in the case of capacity forecasting is due to the fact that the original objective is not differentiable or even continuous, hence cannot be directly used as a loss function: as mentioned in Section III-B, and as a desirable by-product of co-training, `LossLeaP` learns a differentiable version of $f_{\mathcal{M}}$, so that the latter can be used to train the predictor DNN.

Complete results from the controlled environment use cases are summarized in Table I. Across all settings, `LossLeaP` stands out as the best peforming model, or a close second; more precisely, when not yielding the best result, our solution is typically within the variance of the method ranking first. A closer inspection of the exact figures reveals several important observations, as follows.

- The models that perform close to `LossLeaP` are those that involve human intervention, which is needed to define a tailored (and possibly parametrizable) loss function for the specific goal, such as `Manual` or `ALA-moldable`; instead, the training of `LossLeaP` is fully automated.
- The models performing best for some use cases tend to have highly fluctuating performance under other objectives, where they generate poor predictions; instead, `LossLeaP` performs consistently well across all target use cases, which demonstrates its flexibility and generality.
- Compared to the benchmarks, `LossLeaP` produces results with sensibly lower standard deviation, which elicits a more consistent quality of the anticipatory decision.
- Juxtaposing `LossLeaP` with its variants proves that all design elements in Section III-B contribute to the performance of the model, and removing co-training (`Iterative`), noisy exploration (`Noiseless`), or learning rate adaptiveness (`Fixed-rate`) deteriorates results.

Overall, the results obtained in the controlled environments clearly showcase the gain of `LossLeaP` over other methods, in terms of sheer performance and flexibility. Importantly, this is attained while also reducing the need for human intervention.

V. Application Case Studies

In order to assess the performance of `LossLeaP` in more taxing contexts that are closer to real-world IBN activation tasks, we consider two practical case studies, detailed next.

In both analyses, we employ real-world traffic measurements to ensure credibility of the results. The data describes the demands generated in a large metropolitan area, during several consecutive months. The collection was carried out in a production 4G network, using passive measurement probes. Individual IP sessions were mapped to specific ser-

vices using Deep Packet Inspection (DPI) and commercial traffic classifiers deployed by the operator. The data was processed in secure premises by the operator, in compliance with international regulations, and under the supervision of the relevant data protection officers. For our study, we had access to de-personalized aggregates of traffic time series at core network and Edge datacenters, for four video streaming services: Facebook Live, Netflix, Twitch, and Youtube.

A. Reserving Virtual Machines at a core network datacenter

In a first case study, we consider a core network datacenter setting, where we assume that each video streaming service is assigned a dedicated Network Slice Subnet Instance (NSSI). The machine-translated intent involves that the Virtual Infrastructure Manager (VIM) responsible for controlling the datacenter resources must predict the number of Virtual Machines (VMs) that need to be allocated in advance to each NSSI, so as to run the Virtual Network Functions (VNFs) required to serve the demand generated by the corresponding mobile service. Clearly, every VM has an operating cost (*e.g.*, due to power consumption) so it is desirable that only the strictly necessary set of VMs is reserved for each NSSI.

Clearly, we do not have access to information about the actual operating costs of the datacenter, or about the local VM management strategies. This forces us to *emulate* that part of the system behavior to perform our experiments. To that end, we proceed as follows: (i) we define a credible expression $f_{\mathcal{M}}$ for the target management objective; (ii) we use $f_{\mathcal{M}}$ to generate observations $\mathcal{M}_{t+1}$ in response to predictions d_t; and, (iii) we use observations $\mathcal{M}_{t+1}$ as ground-truth measurements to train and test `LossLeaP` and benchmark models. A very important remark is that $f_{\mathcal{M}}$ is only an artifice we adopt to generate measurement-like data, and it is *unknown* to `LossLeaP`, which has only access to the observations.

Formally, let all VMs have equivalent performance, so that each has operating cost κ, and can serve a maximum volume of traffic C_σ that may depend on the slice σ. Also, $\eta \gg \kappa$ is a high penalty triggered in case insufficient VMs are allocated in advance. The observations are computed as:

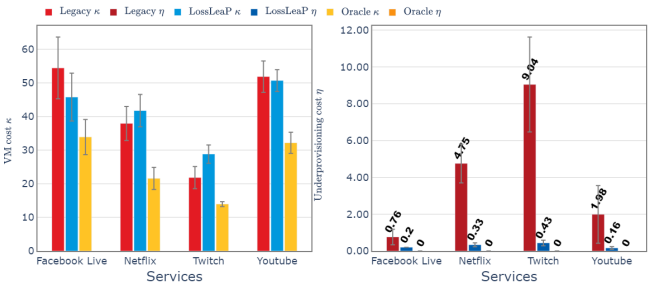
$$\mathcal{M}_{t+1} = f_{\mathcal{M}}(d_t, s_{t+1}) = \eta \cdot \mathcal{R}\left(s_{t+1} - C_\sigma \cdot \left|\lfloor d_t \rfloor\right|\right) + \kappa \cdot \mathcal{R}\left(C_\sigma \cdot \left|\lfloor d_t \rfloor\right| - s_{t+1}\right). \quad (3)$$

This expression basically forces a high handicap if insufficient VMs are allocated, and a milder one for VM overprovisioning.

The VM orchestration takes place at every 5 minutes, which is thus the forecast horizon of the predictor. We feed `LossLeaP` with past traffic information $\bar{s}_t = \{s_{t-N}, \ldots, s_t\}$, with $N = 6$, and let it learn to produce a forecast d_t that minimizes (3), from observations $\mathcal{M}_{t+1}$. We remark that the objective is not linear nor differentiable over the whole domain, as it includes floor and absolute value operators.

We consider two benchmarks for comparative analysis.

- An `Oracle` predictor, which has perfect knowledge of the future, and always allocates the optimal minimum number of VMs to serve the upcoming demand.



(a) VMs reserved (b) Underprovisioning time (%)

Fig. 8: VM reservation for diverse slices. (a) Reserved VMs. (b) Fraction of time when the slice demand cannot be served.

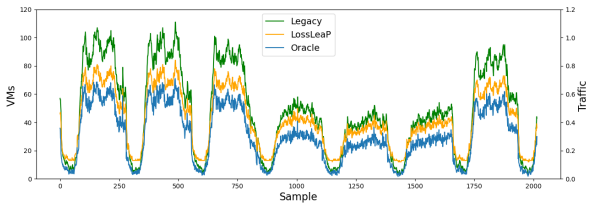


Fig. 9: Example of VM reservation for the Facebook slice.

- A `Legacy` traffic forecasting model, implemented as the predictor part of `LossLeaP`, and trained to minimize an MSE loss. As discussed in Section II, this model requires an additional downstream block in charge of taking the management decision. In this use case, we manually design a VM allocation algorithm that (i) applies an overprovisioning factor to the predicted traffic so as to avoid expensive underdimensioning, and (ii) uses C_σ to compute the number of VMs to serve the inflated demand. Upon extensive tests, we set the overprovisioning factor to 1.6, which enforeces 60% safety margin.

Figure 8 summarizes the performance, when the C_σ and κ are set to 10% and 1% of η. Even when fine-tuned, a decision making policy based on a `Legacy` prediction is substantially less efficient than `LossLeaP`. Our model allocates around the same VMs as `Legacy`, but with an extremely limited underprovisioning that is one or two orders of magnitude lower than that of `Legacy`. The reason is illustrated in Figure 9, for one sample slice: `LossLeaP` anticipates a constant overdimensioning at all times; instead, the solution based on the `Legacy` predictor tends to allocate excess VMs during the high-traffic daylight hours, and does not leave a wide enough safety margin overnight, when it allocates less VMs than `Oracle`, hence not servicing part of the demand. Instead, `LossLeaP` learns that a static overprovisioning factor is not a good strategy to cope with the inherent forecast inaccuracy, and automatically identifies a better loss to minimize the (unknown) expression in (3). By doing so, our model performs in fact fairly close to the `Oracle`.

B. Minimizing video streaming OPEX at the Edge

We now move to a mobile Edge environment, where a set of computational facilities, each serving between 70 and 130 base stations, are located close to the radio access. We assume

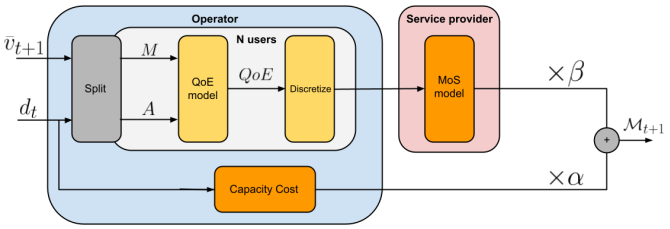


Fig. 10: Emulation pipeline for the monetary OPEX.

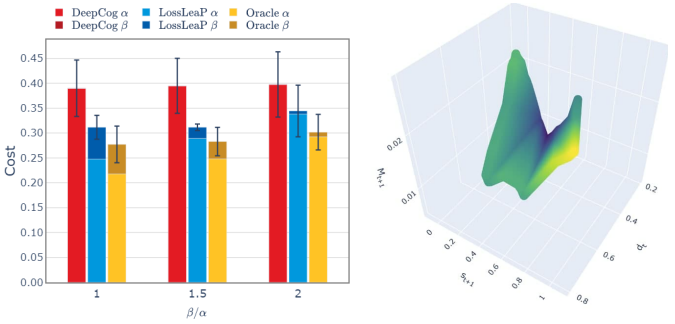
again that each of the four video streaming services has a dedicated NSSI at the Edge facilities. Here, the management intent aims at minimizing the monetary operating expenses (OPEX) associated to running the video streaming slices at the network Edge. This maps to an IBN objective of periodically and preemptively rescaling the compute resources assigned to each NSSI in a facility, to smoothly run the needed VNFs.

As for the previous use case, we have to emulate the ground-truth OPEX, by developing a sensible model that relates OPEX to the system variables. Here, we go a step further in terms of complexity, and, rather than defining a single expression for $f_{\mathcal{M}}$ as in (3), we employ a whole pipeline to mimic the objective. Figure 10 offers a high-level view of the pipeline. First, the slice-level allocated computing capacity d_t and the future traffic to be served $\bar{v}_{t+1}$ are split across users according to a probability distribution that describes the imbalance of traffic generated by different users. The per-user maximum requested (M, from $\bar{v}_{t+1}$) and available (A, from d_t) compute powers are fed to an empirical non-linear model that maps such metrics into a user-level video streaming Quality of Experience (QoE) [31]. The QoE value is then discretized in line with real-world QoE metrics, and passed to a module that converts the QoE into a Mean Opinion Score (MOS). Based on the MOSs of the users, the slice tenant can communicate to the operator if the Service Level Agreement (SLA) has been violated, which entails a monetary fee β per violation. This cost is summed to that generated by the need to reserve the compute resources d_t to accommodate the slice traffic, derived in the bottom block, at a price α per capacity unit.

Interestingly, not only the expression of $f_{\mathcal{M}}$ is now much more involved than before, but it is also *not observable* by the operator in practical cases. Indeed, the VNF Manager (VNFM) that must take the anticipatory decision only has data about the current allocated resources and slice traffic demand from the VIM. In the best case, it may consume QoE information provided by the Network Data Analytics Function (NWDAF) [32] that interfaces with Application Functions (AF), but the MOS part of the pipeline remains unknown. Ultimately, this use case sets forth a scenario where a network manager would not have the necessary system knowledge to manually design a solution to the problem.

We let `LossLeaP` learn the whole pipeline above in a setting where the compute capacity reconfiguration occurs at every 5 minutes. We compare it against two benchmarks, as follows.

- An `Oracle` predictor that knows the future demand and the full $f_{\mathcal{M}}$ pipeline, and returns an optimal allocation.



(a) Overall performance (b) Learned loss

Fig. 11: OPEX performance in the Facebook Live slice case. (a) Overall cost. (b) Loss function learned by `LossLeaP`.

- `DeepCog`, a state-of-the-art capacity predictor [22] that we configure to cope with the problem in the best way possible, by setting its loss function parameter to β/α.

Due to space limitations, we only show results for the Facebook Live slice, in Figure 11a, however performances are homogeneous across services. All costs are relative to that of the minimum static capacity allocation that always services the full demand, and are shown for different β/α ratios. Despite the fact that we feed it with information about the true α and β values, `DeepCog` is still constrained by its unflexible loss function. Instead, `LossLeaP` can autonomously learn a much better loss, which results in a reduced cost closer to the `Oracle` one. Figure 11b offers a glance at the faily complex loss function captured by the loss-learning DNN of `LossLeaP`: even if full knowledge of the pipeline in Figure 10 were available, designing manually devise this shape would be an exacting task. Our approach effectively automates such design, which lays an important stone in the path to full-fledged IBN.

VI. Conclusions

We proposed `LossLeaP`, a first-of-its-kind model that automates the design of loss functions for regression tasks, and, more precisely, prediction problems. Extensive tests in realistic settings and against a wide range of benchmarks demonstrate the viability of our concept, as well as the potential performance gains it can unlock. Our work contributes to advancing the state of the art in automated network management, setting forth an important contribution towards practical IBN systems. It also makes a step forward in the AutoML domain, providing a first solution to an open problem in machine learning.

The authors have provided public access to their code and/or data at https://github.com/alcoimdea/LossLeaP.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101017109 "DAEMON". The authors thank Sachit Mishra for his support with data preparation. Figure 1 has been designed using resources from Flaticon.com.

REFERENCES

[1] ETSI, "Zero touch network and service management (zsm) means of automation," 2018.

[2] C. Janz, N. Davis, D. Hood, M. Lemay, D. Lenrow, L. Fengkai, F. Schneider, J. Strassner, and A. Veitch, "Intent nbi–definition and principles," *Open Networking Foundation, Version*, vol. 2, 2015.

[3] D. Schulz, "Intent-based automation networks: Toward a common reference model for the self-orchestration of industrial intranets," *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 4657–4664, 2016.

[4] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, "Intent-based networking-concepts and definitions," *IRTF draft work-in-progress*, 2020.

[5] 3GPP, "Technical specification group services and system aspects, "tr:28.812 – study on scenarios for intent driven management services for mobile networks, telecommunication management," 2020.

[6] ETSI, "Experiential networked intelligence (eni), "context-aware policy management gap analysis."," 2018.

[7] S. SECTOR, "Focus group on machine learning for future networks including 5g ml5g-i-125-r3."

[8] Y. Wei, M. Peng, and Y. Liu, "Intent-based networks for 6g: Insights and challenges," *Digital Communications and Networks*, vol. 6, no. 3, pp. 270–280, 2020.

[9] F. Xu *et al.*, "Big Data Driven Mobile Traffic Understanding and Forecasting: A Time Series Approach," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 796–805, Sep. 2016.

[10] M. Zhang *et al.*, "Understanding Urban Dynamics From Massive Mobile Traffic Data," *IEEE Transactions on Big Data*, vol. 5, no. 2, pp. 266–278, Nov. 2017.

[11] S. T. Au *et al.*, "Automatic forecasting of double seasonal time series with applications on mobility network traffic prediction," *JSM Proceedings, Business and Economic Statistics Section*, Jul. 2011.

[12] S. Ntalampiras *et al.*, "Forecasting mobile service demands for anticipatory MEC," in *Proc. of IEEE WoWMoM*, Chania, Greece, Jun. 2018, pp. 14–19.

[13] C. Gijón, M. Toril, S. Luna-Ramírez, M. L. Marí-Altozano, and J. M. Ruiz-Avilés, "Long-term data traffic forecasting for network dimensioning in lte with short time series," *Electronics*, vol. 10, no. 10, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/10/1151

[14] M. Z. Shafiq *et al.*, "Characterizing and modeling internet traffic dynamics of cellular devices," in *Proc. of ACM SIGMETRICS*, San Jose, CA, USA, Jun. 2011, pp. 265–276.

[15] R. Li *et al.*, "The prediction analysis of cellular radio access network traffic: From entropy theory to networking practice," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 234–240, Jun. 2014.

[16] C. Zhang *et al.*, "Deep Learning in Mobile and Wireless Networking: A Survey," *arXiv:1803.04311 [cs.NI]*, Mar. 2018.

[17] J. Wang *et al.*, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. of IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[18] A. Y. Nikravesh *et al.*, "An Experimental Investigation of Mobile Network Traffic Prediction Accuracy," *Services Transactions on Big Data*, vol. 3, no. 1, pp. 1–16, Jan. 2016.

[19] C. Zhang *et al.*, "Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks," in *Proc. of ACM MobiHoc*, Los Angeles, CA, USA, Jun. 2018, pp. 231–240.

[20] J. X. Salvat *et al.*, "Overbooking Network Slices Through Yield-driven End-to-end Orchestration," in *Proc. of ACM CoNEXT*, Heraklion, Greece, Dec. 2018, pp. 353–365.

[21] C. Gutterman *et al.*, "RAN resource usage prediction for a 5G slice broker," in *Proc. of ACM Mobihoc*, Catania, Italy, Jul. 2019.

[22] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 280–288.

[23] ——, "Aztec: Anticipatory capacity allocation for zero-touch network slicing," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 794–803.

[24] D. P. Kingma *et al.*, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, Dec. 2014.

[25] P. C. Pendharkar and P. Cusatis, "Trading financial indices with reinforcement learning agents," *Expert Systems with Applications*, vol. 103, pp. 1–13, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418301209

[26] J. W. Lee, "Stock price prediction using reinforcement learning," in *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)*, vol. 1, 2001, pp. 690–695 vol.1.

[27] C. Huang, S. Zhai, W. Talbott, M. B. Martin, S.-Y. Sun, C. Guestrin, and J. Susskind, "Addressing the loss-metric mismatch with adaptive loss alignment," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2891–2900. [Online]. Available: http://proceedings.mlr.press/v97/huang19f.html

[28] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705120307516

[29] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017, pp. 464–472.

[30] R. Koenker and K. F. Hallock, "Quantile regression," *Journal of Economic Perspectives*, vol. 15, no. 4, pp. 143–156, December 2001. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/jep.15.4.143

[31] J. Nightingale, P. Salva-Garcia, J. M. A. Calero, and Q. Wang, "5g-qoe: Qoe modelling for ultra-hd video streaming in 5g networks," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 621–634, 2018.

[32] 3GPP, "Architecture enhancements for 5g system (5gs) to support network data analytics services (3gpp ts 23.288 version 16.4.0 release 16)," 2020.