This is the author's accepted version of the article. The final version published by IEEE is Alan Collet, Antonio Bazco-Nogueras, Albert Banchs, Marco Fiore, "AutoManager: a Meta-Learning Model for Network Management from Intertwined Forecasts" IEEE INFOCOM 2023 - IEEE Conference on Computer Communications, 2023, pp. TBD, doi: TBD.

AutoManager: a Meta-Learning Model for Network Management from Intertwined Forecasts

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

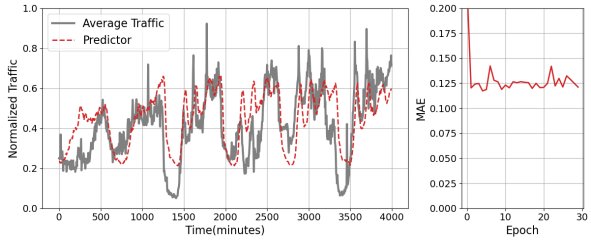
***Abstract*—A variety of network management and orchestration (MANO) tasks take advantage of predictions to support anticipatory decisions. In many practical scenarios, such predictions entail two largely overlooked challenges: (*i*) the exact relationship between the predicted values (*e.g.*, reserved resources) and the performance objective (*e.g.*, quality of experience of end users) is often tangled and cannot be known a priori, and (*ii*) the objective is linked in many cases to multiple predictions that contribute to it in an intertwined way (*e.g.*, resources to reserved are limited and must be shared among competing flows). We present `AutoManager`, a novel meta-learning model that can support complex MANO tasks by addressing these two challenges. Our solution learns how multiple intertwined predictions affect a common performance goal, and steers them so as to attain the correct operation point under a-priori unknown loss functions. We demonstrate `AutoManager` in practical, complex use cases based on real-world traffic measurements; our experiments show that the model produces forecasts that are accurate and tailored to the MANO task in a fully automated way.**

I. Introduction

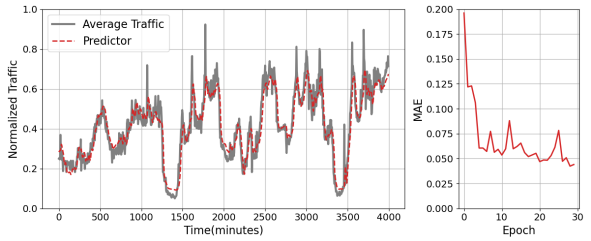
The automation of network management plays a core role in the vision for 6G [1]. The increasing softwarization and programmability of mobile networks, and the redesign of network functions for cloud-native operation lay the foundations to automation paradigms like zero-touch networking and service management (ZSM) [2] and intent-based networking (IBN) [3]. Data-driven models will be important enablers in this ecosystem, and standard-defining organizations (SDO) are integrating machine learning operations (MLOps) into network management and orchestration (MANO) frameworks [4]–[6].

Limits of legacy traffic prediction. Data-driven machine learning models are expected to support in particular anticipatory operations, which are ostensibly more effective than reactive policing and unlock the full benefits of automation [7].

As discussed in Section II, forecasting for MANO is almost invariably tackled with supervised regression models that either (*i*) ignore the relationship between the prediction output and the performance objective, or (*ii*) assume that perfect knowledge of such a relationship is available at model design time. Yet, in practical cases, the relationship between predictions and final performance is not known a priori, and cannot be neglected as it is paramount to an effective anticipatory decision. Examples abound: for instance, forecasts of computing resources allocated to specific network slices affect the Quality of Experience (QoE) of the end users in ways that are hard to estimate in advance [8]; anticipatory associations of Distributed Units (DUs) to Central Units (CUs) in virtualized Radio Access Networks (vRANs) are linked to performance and energy consumption in tangled ways that are best observed only after deployment in real-world systems [9]; or, all IBN use cases inherently require translation of high-level intents into objectives whose relation to the actual MANO decisions are unknown a priori [10].



(a) `LossLeaP`



(b) `AutoManager`

Fig. 1: Performance of (a) `LossLeaP` and (b) `AutoManager` in the toy use case of mean value forecasting. Left: predictions and target average. Right: forecasting error over train epochs.

Learning to predict for MANO. In cases like those above, the data-driven forecasting model shall learn at once (*i*) how its predictions affect the performance goal, and (*ii*) how to optimize such predictions so that the goal is met. In machine learning terminology, this corresponds to the problem of *meta-learning* the loss function that shall drive the training of the model. As detailed in Section II, loss meta-learning is a very recent and active field of study, and no ultimate solution exists for regression tasks such as forecasting.

A challenging and open problem in loss meta-learning that is particularly relevant in the context of MANO is handling *intertwined predictions*, *i.e.*, cases where multiple forecasts contribute to a same goal. Essentially any multiple-input MANO problem falls in this category, including the predictive scheduling of limited capacity across users, the proactive admission control of multiple requesting traffic flows to a network service, the anticipatory allocation of shared resources, or the forecasting of anomalies from joint analysis of different Key Performance Indicator (KPI) streams, just to name a few.

Intertwined predictions are problematic for current solutions for loss meta-learning regressors. We demonstrate the issue via a toy example, where two predictors a and b are fed with past

values of traffic time series y^a and y^b up to time t, and they have to forecast the mean of the two next traffic values, *i.e.*, $(y^a_{t+1} + y^b_{t+1})/2$. While fictional, this is a naive case where the two predictions are intertwined, as their output depends on both flows y^a and y^b. Abiding by the principles above, the expression of the prediction target is not known a priori, and the model must (i) meta-learn that the averaging function is the right loss for the task, and (ii) use it to train the predictors.

Figure 1a shows the performance in the simple task above of a predictor implementing `LossLeaP` [11], a recent solution that laid the foundations for loss meta-learning in MANO, and which represents the current state of the art. The forecast is clearly misaligned with respect to the mean traffic target (left), and results in a poorly converged error (right). The reasons for such poor performance are detailed in Section IV.

Contributions. In this paper, we present `AutoManager`, a forecasting model that supports loss meta-learning in intertwined predictions. As such, and without any prior information about the system, `AutoManager` can learn how multiple predictions affect a common performance goal, and steer them to the correct operation point. Figure 1b shows the performance of our solution for the traffic averaging toy example: now the forecast matches the target, and the error convergence is substantially improved. Overall, the development of `AutoManager` yields the following main contributions.

- We identify significant limitations of the state-of-the-art models for network forecasting, including recent solutions with tailored or meta-learned loss functions, in Section II.
- We propose a new neural network architecture that addresses such limitations via a combination of parallel and cascaded blocks, in Section III. The new model enjoys (i) improved modularity and scalability, and (ii) capability to learn the intertwined relationship between the performance goal and multi-dimensional inputs, as shown in controlled experiments in Section IV.
- We implement and demonstrate our solution with real-world traffic measurements in two practical use cases, *i.e.*, (i) joint anticipatory admission control and resource allocation of network slices, in Section V, and (ii) joint proactive functional split and virtual machine reservation in virtualized radio access networks, in Section VI.

These contributions let `AutoManager` set a new standard in data-driven forecasting for automated MANO, closing important design gaps and contributing to pave the road towards the deployment of machine learning in production networks.

II. Background and related work

Our study relates to and advances proposals for traffic and capacity forecasting, as well as loss meta-learning approaches from the machine learning and networking communities. We next discuss each such class of prior work separately.

Traditional traffic forecasting. Traditional models for traffic forecasting are agnostic to the network management goal, and completely ignore the relationship between the prediction output and the performance objective. These models aim at estimating as accurately as possible the future evolution of the target demand or KPIs that are considered useful to the MANO task, by minimizing legacy error functions such as the Mean Squared Error (MSE) or Mean Absolute Error (MAE). While these models represent the vast majority of the literature [12], they only provide an input to the decision process, which is entirely delegated to a different entity, as shown in Figure 2a.

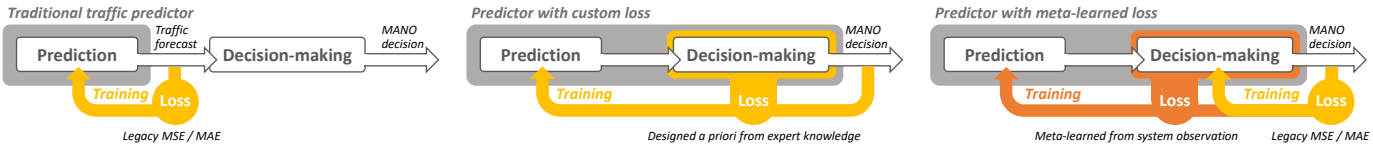
Loss function customization. Making predictors aware of the management goal is in fact very beneficial to MANO. By coupling prediction and decision phases, inherent properties of the forecast (*e.g.*, different accuracy levels of the prediction that depend on the specific input pattern) become available to decision-making, which can then adapt to them.

The integration of the two phases is achieved in machine-learning models via the customization of the loss function used during training. By tuning the shape of the loss in a way that it reflects the MANO objective, the predictor is optimized to produce forecasts that align with the actual management goal. The concept is illustrated in Figure 2b, and results in a model that directly outputs anticipatory MANO decisions, rather than simple traffic predictions. Proposals like `DeepCog` [13] and `CATP` [14] design custom losses that capture monetary costs or that embed domain knowledge as regularization terms. By coupling forecast and decisions, such models yield performance gains above 50% in practical MANO use cases [13], [14].

Loss function meta-learning Approaches based on custom loss functions assume that the relation between the prediction and the ultimate performance goal of the MANO task is fully known and can be described through a suitable (*e.g.*, differentiable and relatively simple) expression. Such perfect knowledge is often not available in practice: to make additional examples to those in Section I, the relation may depend on the many and varied hardware and software solutions deployed in the infrastructure; on information that is only available to the service provider and transparent to the network operator; or, on complex multi-domain interactions that cannot be characterized in advance, even with significant expert knowledge.

In these situations, the loss function can still be tailored to the MANO objective, but not at design time. Instead, the forecasting model shall be able to meta-learn the correct loss during training, along with the usual parameters of the predictor, as exemplified in Figure 2c. Loss meta-learning is an active research topic in the machine learning community. Most solutions aim at inferring a suitable configuration of a predefined, parametrizable loss function [15]–[17], possibly starting from primitive mathematical operators [18]. Yet, in all these cases, the initial loss or its components must still be designed manually, which is not just feasible in MANO settings where the prediction-decision relationship is unknown.

Clean-slate loss meta-learning. What is needed in our case is a strategy that can meta-learn a *clean-slate* loss. Closer to this requirement, the `meta-critic` model [19], [20] employs a trainable task-parametrized loss generator, which however only operates on discrete-space classification or ranking tasks; instead, forecasting is intrinsically a regression problem in MANO tasks, where capacity and resources are very granular and decisions need to be taken on a continuous space.



(a) (b) (c)

Fig. 2: Different approaches to forecasting for MANO. (a) Traditional traffic forecasting that minimizes a MSE or MAE in the prediction of future traffic. (b) Forecasting based on a custom loss that is manually designed to mimic the decision-making problem, so that the output of the model is directly the MANO decision. (c) Forecasting with a meta-learned loss, which is optimized during training so as to represent the decision-making problem, and used in turn to train the predictor.

The only solution for loss-meta learning in regression tasks has been precisely proposed in the context of MANO: `LossLeaP` [11] jointly trains a loss-learning block and the actual predictor so as to output forecasts that abide by the performance goal, which is not known in advance but learned over time. Yet, as illustrated via a toy example in Section I, `LossLeaP` cannot handle intertwined predictions: this is a major limitation in practical network automation cases, where distributed predictions concur to a joint objective. Our solution overcomes this limit, among other advantages.

III. The AutoManager Model

The proposed model, `AutoManager`, is a multiple-output loss-function-agnostic regressor that performs a twofold learning. As set out in the previous sections, (i) it aims at learning to forecast the values of multiple actions that will concurrently optimize a certain global objective. However, as it is oblivious to the objective to optimize, (ii) it must also learn which is the loss function that correctly represents the said objective according to a-posteriori system measurements.

A. Problem Formulation

Formally, let us denote the input space of the predictor as $\mathbb{X}$ and the output space as $\mathbb{Y}$, such that the predictor can be modeled as $f_{\mathbf{W}^p} : \mathbb{X} \rightarrow \mathbb{Y}$, and the multi-dimensional decision taken at time t by the predictor for time $t+1$ can be written as $\hat{\boldsymbol{y}}_{t+1} = f_{\mathbf{W}^p}(\boldsymbol{X}_t)$. Here, $\boldsymbol{X}_t = \{\boldsymbol{x}_{t-T}, \ldots, \boldsymbol{x}_t\} \in \mathbb{X}$ includes observations of the input space for the past T time steps, and $\mathbf{W}^p$ represents the parameters of the predictor. Let us denote by $n_{in} = |\boldsymbol{x}_t|$ and by $n_{out} = |\hat{\boldsymbol{y}}_{t+1}|$ the number of input and output variables, respectively. Note that $\hat{\boldsymbol{y}}_{t+1} = \{\hat{y}^{(1)}_{t+1}, \ldots, \hat{y}^{(n_{out})}_{t+1}\}$ is a vector, and hence the predictor must solve a generic multidimensional forecasting problem. Also, we remark that $\hat{\boldsymbol{y}}_{t+1} \neq \boldsymbol{x}_{t+1}$ in general, *i.e.*, the output is not a direct prediction of the input as in traditional traffic forecasts; instead, each decision $\hat{y}_{t+1}$ is a compound function dependent on the whole set of input variables $\boldsymbol{X}_t$.

The MANO performance cost resulting from the decision taken at time t is denoted by $\mathcal{M}_{t+1} = f_{\mathcal{M}}(\hat{\boldsymbol{y}}_{t+1}, \boldsymbol{v}_{t+1})$, which depends on the prediction $\hat{\boldsymbol{y}}_{t+1}$ itself, and on system variables $\boldsymbol{v}_{t+1}$ that may impact the performance at $t+1$. It is important to recall that the expression of $f_{\mathcal{M}}(\cdot)$ is *unknown a priori*, as it is too complex to characterize or it depends on information not available at model design time. Yet, performance is measurable

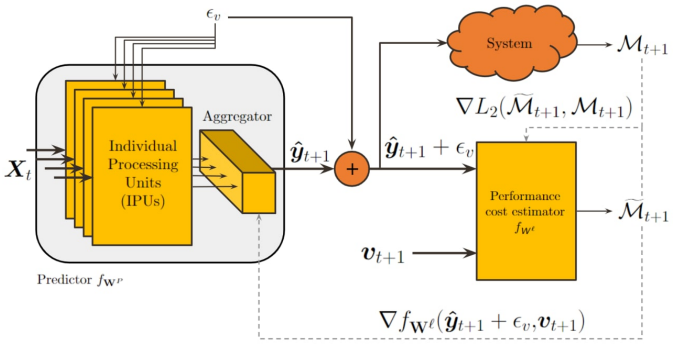


Fig. 3: Architecture and training of the `AutoManager` model.

at time $t+1$, and samples of $f_{\mathcal{M}}(\cdot)$ can be obtained by observing the outcome of $\hat{\boldsymbol{y}}_{t+1}$ on the system.

In order to make correct decisions that align with the MANO objective, the initially uncharted $f_{\mathcal{M}}(\cdot)$ must be first discovered. This means solving the optimization problem

$$\min_{\mathbf{W}^\ell} \; L^2\Big(f_{\mathbf{W}^\ell}(\hat{\boldsymbol{y}}_{t+1}, \boldsymbol{v}_{t+1}), f_{\mathcal{M}}(\hat{\boldsymbol{y}}_{t+1}, \boldsymbol{v}_{t+1})\Big), \tag{1}$$

i.e., identifying an estimator $\tilde{\mathcal{M}}_{t+1} = f_{\mathbf{W}^\ell}(\hat{\boldsymbol{y}}_{t+1}, \boldsymbol{v}_{t+1})$ whose parameters $\mathbf{W}^\ell$ minimize the L^2 loss[1] with respect to the actual cost $\mathcal{M}_{t+1}$. The performance cost estimator from (1) can then be used to steer predictions towards the performance objective, by embedding it into a second optimization problem

$$\min_{\mathbf{W}^p} \; f_{\mathbf{W}^\ell}(f_{\mathbf{W}^p}(\boldsymbol{X}_t), \boldsymbol{v}_{t+1}\big). \tag{2}$$

The solution to (2) is the predictor $f_{\mathbf{W}^p}$ that we seek, which produces a forecast minimizing the expected MANO cost $f_{\mathbf{W}^\ell}$ at time $t+1$ from the input $\boldsymbol{X}_t$ at time t.

B. Model overview

In order to jointly solve the optimization problems in (1) and (2), `AutoManager` adopts a data-driven deep-learning approach that follows the general design for loss meta-learning outlined in Figure 2c. Therefore, similarly to `LossLeaP` [11], our solution models the decision-making part of the MANO task with a dedicated neural network, which is trained to solve the cost estimation problem using observations of the outcome of the predictive decisions. A second neural network implements instead the actual predictor, and it is trained to minimize the

[1]The goal of the estimator is to mimic as closely as possible $f_{\mathcal{M}}(\cdot)$: any loss minimizing the error between $f_{\mathbf{W}^\ell}(\cdot)$ and $f_{\mathcal{M}}(\cdot)$ suits well the purpose.

cost estimated by the first network. `AutoManager` differs from earlier proposals in the implementation of the general two-stage design above, as portrayed in Figure 3 and detailed next.

The *predictor* receives the input $\boldsymbol{X}_t$ and generates the anticipatory decisions $\hat{\boldsymbol{y}}_{t+1}$, thus realizing $f_{\mathbf{W}^p}$. Internally, the predictor is in fact organized into multiple *individual processing units* (IPU), each receiving past values for one of the variables that compose the input: denoting by $\boldsymbol{x}_t = \{x_t^{(1)}, \ldots, x_t^{(n_{in})}\}$ the input space at time t, then the i-th IPU is fed with a time series $\{x_{t-T}^{(i)}, \ldots, x_t^{(i)}\}$. The individual processing units are not connected to each other, and their task is learning the unique temporal correlations that characterize each time series and that are useful to the anticipatory MANO decisions.

The output of all individual processing units is then gathered by a single *aggregator*, which is an integral part of the predictor. The aggregator models how the time patterns of each input variable are intertwined to produce a given performance, and how to exploit such temporal information to produce predictions $\hat{\boldsymbol{y}}_{t+1}$ that optimize the MANO objective.

Finally, the *performance cost estimator* receives the decision $\hat{\boldsymbol{y}}_{t+1}$ as well as the relevant system state $\boldsymbol{v}_{t+1}$ observed at time $t+1$, and outputs the approximate cost determined by the application of such a decision in the system. Hence, the estimator implements $f_{\mathbf{W}^\ell}$: its sole purpose is to meta-learn the loss to train the predictor, and it is not used during inference.

The organization of the predictor represents the main novelty of `AutoManager`, and yields a number of advantages over previous solutions and alternative approaches, as follows.

- The configuration into individual processing units plus one aggregator gives a logical structure to the predictor, breaking down the whole problem of optimizing $f_{\mathbf{W}^p}$ into simpler learning tasks. The IPUs and aggregator, trained via the same gradient descent as explained in Section III-C, improve the efficiency and scalability of the model, *e.g.*, with respect to a large monolithic predictor that receives $\boldsymbol{x}_t$ and outputs $\hat{\boldsymbol{y}}_{t+1}$, as demonstrated in Section IV.
- Individual processing units also grant a higher modularity to the model. In particular, they enable the pre-training of each unit on historical data about their target input variable, as explained in Section III-C, so as to cut training times.
- The `AutoManager` predictor supports multiple-output forecasts optimizing a single compound metric. This allows solving MANO problems with intertwined anticipatory decisions, unlike prior solutions, as mentioned in Section I and later proven in Section IV.
- The proposed model also works with non-time-correlated input variables, which is not the case with other neural architectural constructs, as we will show in Section IV.

Overall, `AutoManager` lets the model converge toward a global shared benefit, rather than having each predictor converging to their own local minimum as in previous solutions [11]. Interestingly, this is a comparable advantage as having a game theory model converge to the global minimum instead of a Nash Equilibrium where each player seeks a local minimum.

C. Detailed implementation

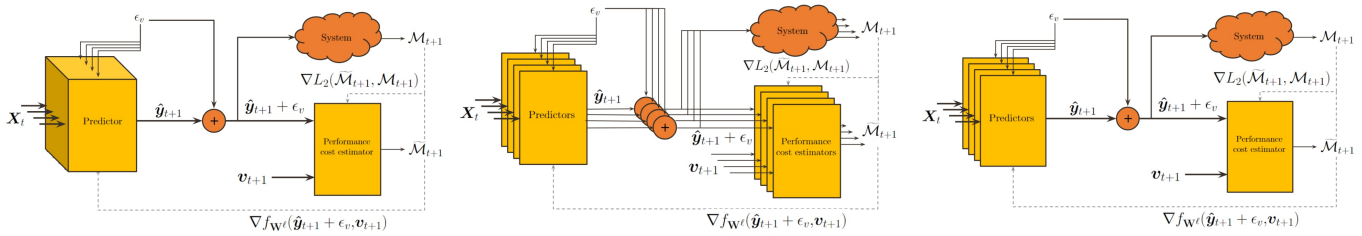
Let us explain the detailed implementation of the proposed model, where we will also distinguish our contributions from the aspects that were already proposed in the literature.

a) Structure adapted to compound relationships: One of the main limitations of previous approaches tackling clean-slate loss meta-learning is their inability to learn involved and intertwined relationships among different variables. Our main contribution to the state of the art in loss meta-learning is the neural architecture in Figure 3, which is able to capture these connections and achieve high performance even in complex MANO use cases such as those in Sections V and VI. Specifically, the parallel architecture of the IPUs allows for improved modularity and scalability, and can accommodate different architectures at each IPU that adapt to the particular variable that the IPU handles. Moreover, the presence of the aggregator block allows us to functionally separate the learning of single-variable predictions and inter-variable correlations.

b) Pre-training of Individual Processing Units: Another main limitation of previous approaches is the lack of optimization for reducing the learning time, which can be naturally higher than in the scenarios in which we know the loss function. To tackle this problem, the neural architecture of `AutoManager` allows to individually pre-train each IPU to minimize $L^2(\hat{x}_{t+1}^{(i)}, x_{t+1}^{(i)})$ from $\{x_{t-T}^{(i)}, \ldots, x_t^{(i)}\}$. This is equivalent to pre-train offline each IPU as a regular time-series predictor, *e.g.*, using historical data about $\boldsymbol{x}_t$. While this pre-training is done with a legacy loss function not aligned with the objective, we find it to speed up learning in actual systems.

c) Joint training: The structure of the model in Fig. 3 allows for *simultaneous training* of all the blocks (IPUs, aggregator, and performance cost estimator). In other words, we can use the same gradient descent iteration to train both the metric learner and the predictor. This approach has been recently proposed in the literature [11], although its application for `AutoManager` is more elaborate: previous works had a cascaded sequence of two blocks (predictor $\rightarrow$ performance cost estimator), whereas we have both cascaded and parallel structures, since we have the sequences IPU $\rightarrow$ aggregator $\rightarrow$ performance cost estimator, but the IPUs are also implemented in parallel and each IPU only receives a subset of the input variables. While this structure makes the joint training far from straightforward, our experiments show that the approach works even in complex use cases, where it drives the model towards correct high-performance operation points.

d) Noise exploration: During training, we incorporate a random zero-mean noise to the output of the aggregator $\hat{\boldsymbol{y}}_{t+1}$ before introducing $\hat{\boldsymbol{y}}_{t+1}$ to the performance cost estimator. This idea, introduced by `LossLeaP` [11] for loss-learning problems and inspired by the common Reinforcement Learning (RL) practice of occasionally taking random decisions to explore unseen states, allows the performance cost estimator to *explore* the whole domain of the loss, hence better map the continuous input-output relationship. In fact, `LossLeaP` treats the variance of this noise as a hyper-parameter that needs to be manually tuned. We improve this by following an *Automated Machine*



(a) bm-monolithic (b) bm-split (c) bm-merged

Fig. 4: Detailed architectures of the (a) bm-monolithic, (b) bm-split and (c) bm-merged benchmarks.

Learning (AutoML) approach [21]: we let the noise variance automatically vary as the training advances. Specifically, the variance decays exponentially, so as to balance exploration and exploitation as per RL best practices. Note that the same noise is also input to the IPUs, so that they can learn its impact on the predicted values at training time, and preserve the correct correlations with $\boldsymbol{X}_t$. After training, when AutoManager is only used for inference, the noise input is set to zero.

e) Internal design of components: While the size of the input (n_{in}) and output (n_{out}) vary according to the target use case, we maintain the following specifications for all the results presented hereinafter. Each IPU consists of a neural network composed of multiple LSTM layers,whereas the aggregator is implemented through a fully connected Multi-Layer Perceptron (MLP) of multiple layers. The exact number of layers and neurons varies depending on the dimensionality of the input and output. These choices follow from the fact that the IPUs take care mostly of characterizing the temporal correlations for each one of the temporal variables, while the aggregator maps the correlation between variables. The performance cost estimator also employs an MLP architecture, as this is the most general model one can adopt to learn completely unknown correlations like those linking $\boldsymbol{y}_{t+1}$ and $\mathcal{M}_{t+1}$.

IV. Controlled Experiments

Before deploying AutoManager in practical MANO use cases, we investigate the performance of the model in a controlled environment. The purpose is to show the advantages of the approach adopted by AutoManager over previous proposals for loss meta-learning for MANO tasks, as well as alternative architectures to that presented in Section III.

We consider the three representative benchmarks below.

- The bm-monolithic approach, shown in Figure 4a, is an obvious way to extend the LossLeaP model to operate with intertwined predictions. It uses a single block with LSTM and MLP layers to implement the predictor, which gets the whole input $\boldsymbol{X}_t$ and outputs all forecasts $\hat{\boldsymbol{y}}_{t+1}$. Thus, the model generalizes the single-input single-output predictor of LossLeaP to a multi-input multi-output one.
- The bm-split design decomposes the multi-dimensional prediction problem along single input and output variables. As illustrated in Figure 4b, this corresponds to running a number $n_{in} = n_{out}$ of LossLeaP instances in parallel, which is the other naive way to generalize that model to intertwined predictions.

TABLE I: Results for the controlled experiments in terms of average MAE, for our model and all benchmarks.

n_{in}	AutoManager	bm-merged	bm-split	bm-monolithic
2	**0.024±0.012**	0.101±0.081	0.119±0.094	0.067±0.042
4	**0.023±0.010**	0.095±0.073	0.116±0.089	0.047±0.026
6	**0.021±0.007**	0.098±0.076	0.090±0.073	0.043±0.026
12	**0.012±0.003**	0.093±0.079	0.088±0.096	0.038±0.024

- The bm-merged architecture is an intermediate solution in between the previous two benchmarks. As depicted in Figure 4c, it employs multiple parallel predictors that feed a single performance cost estimator during training.

The benchmarks above cover different strategies to extend the state-of-the-art LossLeaP model to multi-dimensional settings. In addition, they can be seen as simplified versions of AutoManager: indeed, bm-monolithic collapses the internal IPU structure of the predictor in Figure 3 into gathered LSTM layers; bm-merged removes instead the aggregator. Hence, the analysis of these approaches provides an ablation study for our model. Finally, we remark that both bm-split and bm-merged implicitly assume that $n_{in} = n_{out}$ and that $\tilde{y}_{t+1}$ can be inferred from x_t only; both models lose generality with respect to AutoManager, which removes such assumptions.

We test all models with the simple toy example introduced in Section I, where the unknown objective is producing an average of all inputs in the next time slot. Formally, using the notation introduced in Section III, the predictor shall output a scalar[2] $\tilde{y}_{t+1} = \frac{1}{n_{in}} \sum_{i=1}^{n_{in}} x_{t+1}^{(i)}$. This is an unsophisticated problem where predictions are intertwined[3], as the output depends on forecasts of all inputs. While it does not resonate with any practical MANO task, this use case offers a way to experiment with the different models in straightforward settings.

Results are summarized in Table II. The decoupling of each output $\tilde{y}_{t+1}^{(i)}$ from all inputs $\boldsymbol{X}_t$ in bm-split and bm-merged creates a clear problem with both models, with errors that are around $4\times$ higher than those attained by AutoManager. This proves, in particular, the key role of the aggregator in combining the contribution of each input and making sure that each IPU is informed of its role towards achieving the objective. For the bm-monolithic model, treating all the input time series as a single vector reduces a lot performances from the LSTM layers. The MAE in the toy use case is more than doubled

[2]The prediction of bm-split and bm-merged is n_{in}-dimensional, hence each $\tilde{y}_{t+1}^{i}$ shall match the same expression in the main text.

[3]We pay attention to input uncorrelated time series, so that their mean cannot be simplified as $\boldsymbol{y}_{t+1} = K\boldsymbol{X}_t$, which would artificially ease the task.

with `bm-monolithic` with respect to our model, showcasing the importance of parallel IPUs to manage independently the input time series that are not naively correlated.

Overall, `AutoManager` yields largely superior performance over the state of the art and competitor architectures even in a simple toy example. In the remainder of the paper, we explore how `AutoManager` scales to more complex MANO use cases.

V. Use Case I: Overbooking of Network Slices

Multi-tenancy is a defining paradigm in 5G systems, which are expected to fully support network slicing providing flexible policing and strong guarantees to heterogeneous services run by different tenants. The anticipatory allocation of isolated and customized resources to individual slices is a key functionality to ensure that the Quality of Experience (QoE) that each tenant request is met. This functionality also offers new opportunities for operators to optimize their revenues: for instance, *slice overbooking* takes advantage of slice demand multiplexing to reduce resource utilization while fulfilling (a priori unknown) QoEs [22]. Inspired by the overbooking approach, we consider a case where anticipatory admission control (AC) and resource reservation (RR) of network slices must be steered to optimize QoE-based revenues for the network operator.

A. MANO objective

We consider that the QoE level determines certain monetary revenues and costs for the network operator, as follows. A Service-Level Agreements (SLA) sets the performance target of a given network slice in terms of the requested end-user QoE, and specifies the amount offered by the tenant for implementing the slice. The SLA also defines how such an amount is reduced for lower QoE levels, including an economic fee for the operator in case the QoE falls below a minimum acceptable threshold.

Formally, let us denote the traffic demand generated by slice $s \in \mathcal{S}$ at time t as $d_s(t)$ and its peak traffic demand as D_s, and let $c_s(t)$ be the amount of resources that the operator allocates to such a slice. We further denote the normalized demand and allocation as $\bar{d}_s(t) \triangleq \frac{d_s(t)}{D_s}$ and $\bar{c}_s(t) \triangleq \frac{c_s(t)}{D_s}$, respectively.

In case the slice is accepted, the tenant pays for the implementation of the slice an amount $R_s(t) = \gamma d_s(t)$, which is proportional to the traffic to be accommodated by some constant γ factor (in \$/byte). However, the amount $R_s(t)$ is paid in full only if a target end-user QoE level is attained; if instead the end users suffer from lower QoE, the operator incurs into a proportionally reduced payment by the tenant.

In our experiments, we assume that the QoE requirement is met if $\delta_s(t) \triangleq \bar{d}_s(t) - \bar{c}_s(t) \leq 0$, *i.e.*, enough resources are reserved by the operator to serve the whole traffic demand of the slice. Otherwise, the QoE (hence the revenues for the operator) drop according to a sigmoid function of $\delta_s(t)$, as recommended by standard models [23] in revenue management and prospect theory [24]. Below a threshold QoE, the sigmoid becomes negative, meaning that the operator receives no payment but must start paying a fee to the tenant. If then the QoE further falls below a minimum acceptable value, the fee jumps to a large amount $K_s(t) = \beta R_s(t)$ that is proportional to the requested resources for the slice. The resulting QoE-based cost of the accepted slice for the operator is

$$\mathrm{QoE}(\delta_s(t)) \triangleq \Big(\tfrac{K_s(t)}{2}\big(\tanh(\alpha\delta_s(t)+\beta)+1\big) - R_s(t)\Big), \quad (3)$$

where α, β are constant that determine how the sigmoid function exactly scales with the underprovisioning $\delta_S(t)$. Besides the revenue from the SLA, the total economic advantage of the operator is affected by two other elements. On the one hand, the overprovisioning of the reserved resources determines an increase of operating expenses (OPEX): therefore, whenever $\delta_s(t) < 0$, the operator incurs an OPEX penalty $-\gamma\delta_s(t)$ that grows linearly with the quantity of unnecessarily reserved resources. On the other hand, committing to allocate more resources than those available in the network infrastructure leads to a global performance drop and possible service outages, with a huge cost M in terms of, *e.g.*, customer churn.

The final performance cost associated to the slice MANO is

$$\mu = \sum_{s\in\mathcal{S}} \Big(\mathbb{1}\big(\bar{c}_s(t)\big)\cdot \mathrm{QoE}(\delta_s(t)) - \mathbb{1}\big(-\delta_s(t)\big)\cdot\gamma\delta_s(t)\Big) + M\cdot\mathbb{1}\Big(\big(\sum\nolimits_{s\in\mathcal{S}} c_s(t)\big) - C\Big), \quad (4)$$

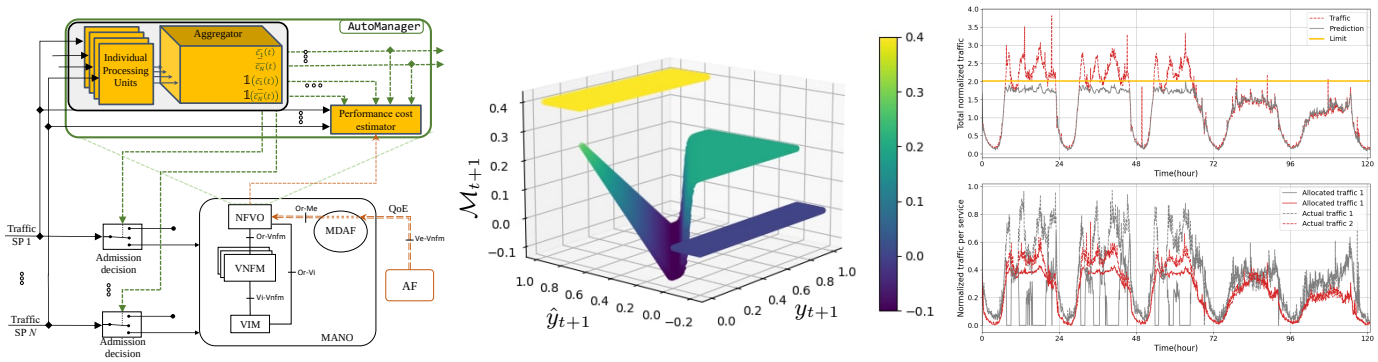
where C is the total capacity of the system, and $\mathbb{1}(\cdot)$ is the step function that takes value one if the argument is less than 0, and value one otherwise: thus, $\mathbb{1}\left(\bar{c}_s(t)\right)$ takes value one only if the slice is admitted and resources are allocated to it.

B. Solution based on `AutoManager`

A key observation is that, in practice, *the network operator cannot analytically know the relation between the slice resource allocation and the QoE of end users of the specific tenant*. This is a very complex function that depends on many operational parameters and whose characterization depends on application-level data (*e.g.*, user feedback) that the operator does not have. Therefore, (3) is not known a priori, in both its whole formulation and specific parametrization of α and β. In turn, this makes it impossible for the operator to model the whole performance cost in (4) in advance.

In this scenario, the operator can apply `AutoManager` to address the MANO task of deciding (i) which slices to accept and (ii) how many resources to reserve to accepted slices at the start of each orchestration interval. Indeed, once deployed in the system, `AutoManager` can learn the expression of the performance cost in (4), and optimize the prediction of $\bar{c}_s(t)$ to minimize such cost. Note that the value of $\bar{c}_s(t)$ determines both the slice admission control, via $\mathbb{1}\big(\bar{c}_s(t)\big)$, and the allocation of resources, which match its value if positive.

The detailed integration of `AutoManager` in the network architecture for this use case is outlined in Figure 5a. Our model sits in the Network Function Virtualization Orchestrator (NFVO) of the MANO framework, as proposed by current standards [6]. It can then leverage the Management Data Analytics Function (MDAF) [25] to access information exposed via the Application Function (AF) [26]: based on the operator-tenant agreement, such information can include live QoE measurements and slice revenue data, which let `AutoManager`



(a) (b) (c)

Fig. 5: Use case I design and results. (a) Implementation of `AutoManager` in the network architecture. B) Sample of the final performance cost, when a single slice $s = \mathcal{S}$ is present in the network. C) Illustration of the `AutoManager` operation.

observe the results of its decisions on the QoE and costs. Our model is then in a position to learn from experience the overall cost in (4), and forecast $\bar{c}_s(t), \forall s \in \mathcal{S}$ to minimize it.

The meta-learning task is in fact not trivial. We provide an intuition of the complexity of the problem in Figure 5b, which illustrates the expression in (4), in the naive case of a single slice. Despite the fact that we are considering the simplest version possible of the cost, and the only one that can be represented in only three dimensions, the shape of the relationship between the MANO objective and the prediction is tangled and also highly sensitive to the value of the input traffic demand. Scaling to realistic. multidimensional versions of the cost thus implies very strong meta-learning capabilities.

C. Benchmark

We compare our solution against the only existing solution to date for loss-meta learning in regression tasks, *i.e.*, `LossLeaP` [11]. However, as shown in Section IV, direct extensions of `LossLeaP` to multi-dimensional cases do not perform well even in toy intertwined prediction scenarios. Thus, our benchmark is built upon `LossLeaP` but incorporates an optimization program to assist it in the MANO task.

In particular, we consider that each slice $s \in \mathcal{S}$ is handled separately by one `LossLeaP` instance, which is responsible for the forecast of the resources to be reserved for a specific slice s in case it is accepted, denoted by $c'_s(t)$. The admission control is instead handled apart, and formulated as an optimization problem solved starting from all $c'_s(t)$ predictions. The problem aims thus at setting the binary admission control variables, $x_s(t)$, which take value one if slice s is accepted. Formally,

$$\max_{x_s(t)} \sum_{s \in \mathcal{S}} R_s(t) x_s(t) \quad \text{s.t.} \sum\nolimits_{s \in \mathcal{S}} c'_s(t) x_s(t) \leq C. \qquad (5)$$

This is in fact the well-known and NP-hard Knapsack problem (KP). Hence, we name the benchmark `LossLeaP-KP`. The individual `LossLeaP` instances are then trained in a similar way as presented in Section V-B, but using observations of the QoE and costs for the relevant slice s only. Clearly, in this case the meta-learning model takes care only of the resource reservation part of the whole MANO task, and it is thus less general and flexible than the solution based on `AutoManager`.

D. Results

We run experiments with real-world mobile traffic of 12 different applications that span video streaming, social networking, cloud and web browsing service categories. The data was recorded at the level of over 400 base stations in a production infrastructure, and described the demands generated by several millions of subscribers in a large urban region. We aggregate the measurements over clusters of 30 base stations computed using the KaFFPa heuristic [27], so as to simulate the traffic experienced at an Edge datacenter. The joint AC-RR task is then solved at each Edge datacenter separately, assuming that each application is requesting a dedicated slice s.

A qualitative illustration of the output of `AutoManager` in this use case is in Figure 5c. The top plot shows how the total demand of all slices (dotted red) exceeds the capacity of the Edge datacenter (gold) at some moments in time: there, our model learns to keep the overall reserved capacity for accepted slices (gray) below the limit. Otherwise, `AutoManager` offers a very precise forecast of the resources to be reserved to accommodate the whole demand. The bottom plot focuses instead on two slices (gray and red), showing their requested capacity (solid) and the allocated resources by our solution (dotted). During periods of low demand, after hour 72, both requests are precisely predicted and fully served. When capacity is scarce, before hour 72, the gray slice is often rejected (with zero reserved resources). Also, the resources allocated to the red slice to not match anymore the request, but are slightly lower: here, `AutoManager` learns and exploits the sigmoid function in (3), which entails a small penalty if the QoE of the end users is reduced only slightly. Indeed, it is more profitable for the operator to gain a bit less on the red slice, and free up space for additional tenants.

It is by precisely by learning the metric accurately and then playing with the impact of the reserved resources on the QoE and monetary costs that `AutoManager` helps the operator benefit from overbooking and maximize its revenues. A quantitative demonstration is in Table II, which reports the

TABLE II: Performance summary for use case I. We report the final *Gain* with *Percent* increase over the benchmark, and average number of slices *Admitted*. Slices vary from 2 to 12.

	AutoManager			LossLeaP-KP	
Slices	Gain	Percent	Admitted	Gain	Admitted
2	**0.0549**	(27.67%)	**1.00**	0.0430	1.00
4	**0.1816**	(12.09%)	**3.18**	0.1620	2.60
6	**0.3363**	(26.38%)	**5.71**	0.2661	4.36
8	**0.3725**	(6.55%)	**6.41**	0.3496	4.83
12	**0.4621**	(36.96%)	**10.99**	0.3374	8.94

final performance figures for our solution and the benchmark, when increasing the number of slices in the system. We focus on non-trivial periods where the MANO task plays a role, *i.e.*, when the slice demands exceed the capacity available at each Edge datacenter. Overall, `AutoManager` lets the operator effectively use slice overbooking to increase its revenues by up to 36% when all 12 slices are present, and to admit 2 slices more than the benchmark on average. We recall that it does so without any prior knowledge of the system or of the objective, which shows its meta-learning capabilities, and the practical advantages entailed by that in solving the MANO task.

VI. Use Case II: Energy-prudent vRAN management

In the second use case, we focus on virtualized RAN (vRAN) management, where the operator must decide on the functional split of the PHY/MAC function pipeline between Distributed Units (DU) residing closer to the radio access, and more powerful Central Units (CU) towards the Edge [28]. We assume that the primary goal of the operator is ensuring the RAN sustainability, by maximizing its energy efficiency [29].

A. MANO objective

Let us assume a specific scenario where N DUs handle the aggregated traffic generated from a set of Remote Units (RU) each. The DUs are associated to a single CU through high-speed mid-haul connectivity. Processing each PHY/MAC function has a different energy cost at the DUs and CU. The energy consumption at each DU is directly proportional to the traffic volume that it has to handle at a given time, since the equipment must be always active. Conversely, the CU runs a number of physical machines (PMs) that can be dynamically turned on/off based on the incoming demand by the DUs [29]. All PMs are identical, and thus have the same energy model [30]: activating a new PM incurs a *start-up* cost, then the energy consumption increases proportionally to the computing load of the PM[4].

Given the traffic demand at one DU, different functional splits entail diverse computational loads at the DU and CU, as they move PHY/MAC functions from the former to the latter. The MANO objective is then deciding, for each DU and decision time, what functional split shall be used to minimize the total energy cost of the vRAN system. It is important to stress that, since this is a preemptive decision, it must be based on a forecast of the mobile data traffic that the DU will observe during the next decision interval.

[4]We assume that the increment of energy per traffic unit is lower at PMs than at DUs, as the CU feature more expensive and scalable equipment.

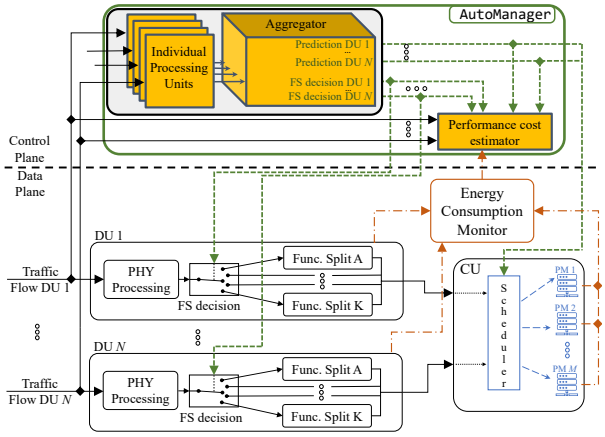


Fig. 6: Implementation of `AutoManager` in use case II.

Once those decisions have been taken, the CU will only activate the minimum required number of PMs to serve the total incoming demand from all DUs. For that, at the start of each decision time, the CU receives the estimated traffic and functional split per DU, and applies a bin-packing [31] near-optimal heuristic algorithm to calculate the minimum amount of PMs required and the associated assignment of PMs to DUs.

B. Solution based on `AutoManager`

We first remark that, in practice, *the network operator does not know the policy on PM activation decided by the cloud provider hosting the CU, or the exact power consumption behavior of the PMs and DUs equipment*. Still, the operator must take functional split decisions and reserve compute capacity in the CU without such information. In addition, the fact that the PM activation decision depends on all DUs sets the stage for a clear instance of intertwined predictions under unknown objective.

In this use case, we can use `AutoManager` to implement the network function that takes care of both (i) deciding on the functional split at each DU, and (ii) forecasting the traffic at each DU used to request compute capacity at the CU. We can see in Figure 6 how the detailed model of `AutoManager` from Figure 3 is integrated in the control plane. In this case, the input to `AutoManager` is composed of N different traffic flows, one for each one of the DUs. `AutoManager` receives the past samples of these flows, and it outputs two sets of values: the first set is a variable per DU that states the specific functional split that is selected, and the second set contains the amount of processing from each DU required at the CU as result of the selected functional split and predicted load. This decision is then implemented and the energy consumption is measured a posteriori, so that it can be fed back to `AutoManager` to train the algorithm and learn the appropriate loss function.

C. Benchmarks

We compare the performance of `AutoManager` in the use case above versus that of two benchmarks. The first one, named `fullDU`, consists in selecting the functional split that makes the DUs handle all the processing of their corresponding RUs; the second benchmark, named `fullCU`, is the case in which all

TABLE III: Performance summary for use case II. Mean power consumption (W) at all DUs and the CU, and increase incurred by the benchmarks with respect to `AutoManager`.

DUs	`AutoManager`	`fullDU`		`fullCU`	
		Power	Increase	Power	Increase
2	**138.4**	180.3	(30.3%)	162.1	(17.1%)
4	**339.2**	365.6	(7.8%)	415.2	(22.4%)
6	**496.8**	542.7	(9.2%)	566.8	(14.1%)
8	**624.4**	663.7	(6.3%)	682.4	(9.3%)
10	**824.7**	861.9	(4.5%)	886.6	(7.5%)

the processing load allowed by the lowest functional possible is transferred to the CU. These two extreme policies must be fed with forecasts of the traffic load at each DU. Since these are traditional predictions of mobile traffic, we employ individual legacy LSTM neural networks [32] to anticipate the load of each DU in the next decision interval. The use of more complex models, such as meta-learning ones, is not necessary for such an obvious prediction task.

We choose these simple approaches because deriving the optimal functional split is not feasible: we would need to solve an extremely complex optimization problem, where the bin-packing algorithm running at the CU, which is itself a NP-hard problem [31], is only a subset of the overall MANO objective.

D. Results

We run tests based on the same measurement data presented in Section V-D. In this case, we assume that a variable number of DUs, from 2 to 10 are deployed in the region, and are assigned to a single CU. Each DU aggregates the total traffic generated by all applications at a disjoint subset of RUs, which we map to base stations. The operator predicts the upcoming traffic at each DU and recomputes the associated functional split at every 5 minutes. The energy consumption of each PM is based on real-world server specifications [33], such that a PM has an energy consumption is 80 W when idle, which linearly grows to 200 W at full load of 200 Mbps.

A view of the overall results of the experiment is provided in Table III, where we list the mean power consumption in W, aggregated over all DUs and the CU. The figures clearly show how `AutoManager` learns the energy cost of different functional splits, unravelling the complex relationship between the anticipatory decision variables and the power consumption at the CU. As a result, `AutoManager` takes MANO decisions on the split and requested resources at the CU that yield a reduction of the mean power consumption in the 4%–30% range with respect to the two benchmarks. It is worth noting that the fact that the performance gain of our solution diminishes with the number of DUs is not a sign that `AutoManager` is not scalable; rather, it is due to the maximum gain being bounded between the power consumption of the PMs used by the `fullDU` model and that one PM less. Then, a larger set of DUs implies that more PMs are required at the CU, and that the maximum gain is inherently reduced.

We also provide details on the temporal variance of such results in the top plot of Figure 7. The lower power consumption granted by our solution is steady over the wide fluctuations of mobile demands over consecutive days. Also, the performance

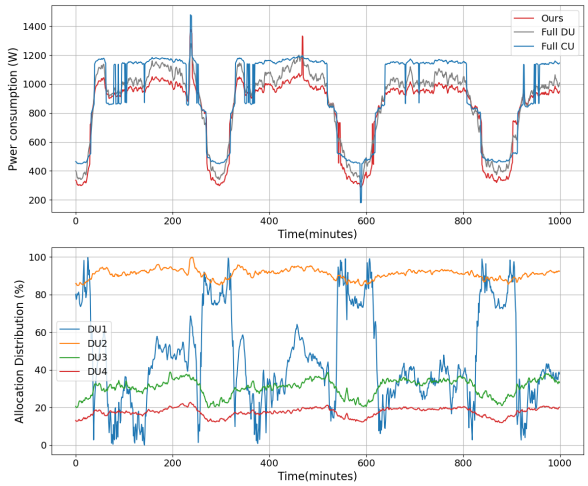


Fig. 7: Temporal detail of the performance of `AutoManager` and the `fullDU` and `fullCU` benchmarks in use case II.

of `AutoManager` is more stable: *e.g.*, the `fullCU` approach determines jumps in the power consumption, as minimum traffic variations can trigger the (de-)activation of an extra PM for a very short amount of time.

The bottom plot in Figure 7 illustrates instead the evolution of the CU resource reservation requests issued by `AutoManager` over time, for a sample of 4 DUs. The abscissa represents the percentage of the total computing load arriving at the i-th DU that is moved to the CU for processing. We observe how `AutoManager` redistributes the load of the DUs to a different extent over time, dynamically adapting the traffic fluctuations. In particular, the model acts on the decision at DU 3 to reach the optimal operation point; when more demand is moved from DU 3 to the CU, the PM resources requested by the other three DUs are slightly reduced to ensure the load of the CU stays at the correct level that minimizes the power consumption.

VII. Conclusions

We have proposed a loss meta-learning approach for time series forecasting that handles complex MANO tasks where multi-dimensional decisions affect a performance objective in a way that cannot be characterized a priori. Our model is general and sets a new standard for data-driven anticipatory prediction that can find application in many MANO use cases, two of which have been demonstrated in the paper.

The authors have provided public access to their code and/or data at https://github.com/nds-group/AutoManager.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements no.101017109 "DAEMON", from the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D project no.TSI-063000-2021-52 "AEON-ZERO", and from the Regional Government of Madrid through the grant 2020-T2/TIC-20710 for Talent Attraction.

REFERENCES

[1] The 5G Infrastructure Association, "European Vision for the 6G Network Ecosystem," 2021.

[2] 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.

[3] ETSI, "GS ZSM 001 V1.1.1 – Zero-touch network and Service Management (ZSM); Requirements based on documented scenarios," 2019.

[4] ITU-T, "Recommendation – Architectural framework for machine learning in future networks including IMT-2020," 2019.

[5] ONAP, "Technical Specification – AI/ML Workflow Description and Requirements v01.02.02," 2020.

[6] ETSI, "GR NFV-IFA 041 – Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Report on enabling autonomous management in NFV-MANO," 2021.

[7] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[8] S. Schwarzmann, C. C. Marquezan, R. Trivisonno, S. Nakajima, V. Barriac, and T. Zinner, "Ml-based qoe estimation in 5g networks using different regression techniques," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[9] H. Gupta, M. Sharma, A. Franklin A., and B. R. Tamma, "Apt-ran: A flexible split-based 5g ran to minimize energy consumption and handovers," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 473–487, 2020.

[10] 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.

[11] A. Collet, A. Banchs, and M. Fiore, "LossLeaP: Learning to predict for intent-based networking," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1–10.

[12] W. Jiang, "Cellular traffic prediction with machine learning: A survey," *Expert Systems with Applications*, vol. 201, p. 117163, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742200553X

[13] 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.

[14] L. A. Garrido, P.-V. Mekikis, A. Dalgkitsis, and C. Verikoukis, "Context-aware traffic prediction: Loss function formulation for predicting traffic in 5g networks," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[15] L. Wu, F. Tian, Y. Xia, Y. Fan, T. Qin, J. Lai, and T.-Y. Liu, "Learning to teach with dynamic loss functions," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 6467–6478.

[16] J. T. Barron, "A general and adaptive robust loss function," 2019.

[17] Q. Liu and J. Lai, "Stochastic loss function," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 4884–4891, Apr. 2020.

[18] H. Li, T. Fu, J. Dai, H. Li, G. Huang, and X. Zhu, "Autoloss-zero: Searching loss functions from scratch for generic tasks," 2021.

[19] F. Sung, L. Zhang, T. Xiang, T. M. Hospedales, and Y. Yang, "Learning to learn: Meta-critic networks for sample efficient learning," *arxiv*, vol. abs/1706.09529, 2017.

[20] W. Zhou, Y. Li, Y. Yang, H. Wang, and T. M. Hospedales, "Online meta-critic learning for off-policy actor-critic methods," 2020.

[21] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[22] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proc. Int. Conf. on Emerging Networking EXperiments and Technologies (CoNEXT*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 353–365. [Online]. Available: https://doi.org/10.1145/3281411.3281435

[23] Z. Tang, F. Zhang, X. Zhou, W. Jia, and W. Zhao, "Pricing model for dynamic resource overbooking in edge computing," *IEEE Transactions on Cloud Computing*, 2022, early access.

[24] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," in *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, 2013, pp. 99–127.

[25] 3GPP TS 28.533 v16, "Management and Orchestration of Networks and Network Slicing; Management and Orchestration Architecture (Rel. 16)," Jun. 2019.

[26] 3GPP TS 29.517 v16.2.0, "5G System; Application Function Event Exposure Service; Stage 3 (Rel. 16)," Mar. 2020.

[27] C. Marquez *et al.*, "How Should I Slice My Network?: A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," in *Proc. of ACM MobiCom*, New Delhi, India, Nov. 2018, pp. 191–206.

[28] L. M. P. Larsen, A. Checko, and H. L. Christiansen, "A survey of the functional splits proposed for 5g mobile crosshaul networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2019.

[29] R. Singh, C. Hasan, X. Foukas, M. Fiore, M. K. Marina, and Y. Wang, "Energy-efficient orchestration of metro-scale 5g radio access networks," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[30] P. Soto, D. De Vleeschauwer, M. Camelo, Y. De Bock, K. De Schepper, C.-Y. Chang, P. Hellinckx, J. F. Botero, and S. Latré, "Towards autonomous VNF auto-scaling using deep reinforcement learning," in *International Conference on Software Defined Systems (SDS)*, 2021, pp. 01–08.

[31] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1973.

[32] 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.

[33] S. Rawas, "Energy, network, and application-aware virtual machine placement model in SDN-enabled large scale cloud data centers," *Multimedia Tools and App.*, vol. 80, no. 10, pp. 15 541–15 562, 2021.