AZTEC+: Long- and Short-Term Resource Provisioning for Zero-Touch Network Management

Sergi Alcalá-Marín, Dario Bega, *Member, IEEE*, Marco Gramaglia, Albert Banchs, *Senior Member, IEEE*, Xavier Costa-Perez, *Senior Member, IEEE*, and Marco Fiore, *Senior Member, IEEE*

***Abstract*—In the past few years, network infrastructures have transitioned from prominently hardware-based models to networks of functions, where software components provide the required functionalities with unprecedented scalability and flexibility. However, this new vision entails a completely new set of problems related to resource provisioning and the network function operation, making it difficult to manage the network function lifecycle management with traditional, human-in-the-loop approaches. Novel zero-touch management solutions promise autonomous network operation with limited human interactions. However, modeling network function behavior into compelling variables and algorithm is an aspect that such solutions must take into account. In this paper, we propose AZTEC+, a data-driven solution for anticipatory resource provisioning in network slicing scenarios. By leveraging a hybrid and modular deep learning architecture, AZTEC+ not only forecasts the future demands for target services but also identifies the best trade-offs to balance the costs due to the instantiation and reconfiguration of such resources. Our experimental evaluation, based on real-world network data, shows how AZTEC+ can outperform state-of-the-art management solutions for a large set of metrics.**

***Index Terms*—Mobile networks, slicing, resource provisioning, zero-touch management, deep learning, traffic prediction.**

I. Introduction

LEVERAGING on the lessons learned from the 5G design, implementation, and deployment, one of the most important characteristics that shall be considered when envisioning the next generation of mobile network architecture is the overall sustainability of the system. Energy and resource efficiency play a crucial role in the aspects that impact this area. In particular, always operating the network with the correct amount of resources, including cloud resources and spectrum resources, can reduce CAPEX and OPEX, yielding an increase in the overall energy efficiency of the system.

Zero-touch resource allocation. In this context, a prominent difficulty is resource management. Isolation of resources across different network slices inherently increases network capacity requirements [1], and a dynamic, preemptive, and efficient allocation of resources to slices is key to keeping efficiency under control in sliced networks [2]. The rapid fluctuations in service demands, as well as the size and complexity of the slicing ecosystem, make legacy reactive, human-driven approaches to resource management inadequate in the emerging context. In sliced networks, automated decisions on resource assignment shall be taken by network orchestrators. By running dedicated Artificial Intelligence (AI) and Machine Learning (ML) solutions [3], network orchestrators are expected to enable the vision of *zero-touch networks* [4], *i.e.*, fully self-operating communication infrastructures whose standardization is already ongoing [5].

Current state-of-the-art solutions for capacity forecasting in network slicing take into account the costs due to (*i*) the allocation of unnecessary resources that go unused, and (*ii*) the insufficient provisioning of resources that cannot accommodate the demand and lead to violations of the Service-Level Agreements (SLA) with the slice tenant. Hence, they aim to minimize overprovisioning while avoiding SLA violations.

Resource instantiation and reconfiguration. Limiting the problem to the simple trade-off above implicitly assumes that resource instantiation and reconfiguration occur at no cost. While this may hold for some types of resource (*e.g.*, CPU time within the same bare metal machine), it is not generally valid for slice resource management scenarios. Instantiation and reconfiguration costs are capital in NFV technologies that enable the *cloudification* of the access and core networks by entrusting many network functions to Virtual Machines (VMs) running in data centers. Examples include baseband processing in Cloud Radio Access Networks (C-RAN) [6], interconnection functionalities towards the external packet networks through the User Plane Function (UPF) [7], or central office operations [8].

In all the above cases, resource instantiation does not take place for free: VM boot times in prominent public

Received 30 July 2024; revised 6 April 2025; accepted 9 June 2025. Date of publication 18 June 2025; date of current version 7 October 2025. This work has been partially supported by the ORIGAMI project, which has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme with Grant Agreement No. 101139270 and by the 6G-IRONWARE project (CNS2023-143870) funded by MICIU/AEI /10.13039/501100011033 and the EU NextGenerationEU/PRTR. The work of the University Carlos III of Madrid has also been funded by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D 6G-CLARION project, and by the TrialsNet Project, which has received funding from the SNS JU under the European Union's Horizon Europe research and innovation programme with Grant Agreement No. 101095871. The associate editor coordinating the review of this article and approving it for publication was R. J. Durán Barroso. *(Corresponding author: Sergi Alcalá-Marín.)*

Sergi Alcalá-Marín, Marco Gramaglia, and Albert Banchs are with the Department of Telematics Engineering, University Carlos III of Madrid, 28911 Madrid, Spain (e-mail: sergi.alcala@imdea.org).

Dario Bega is with Nokia Bell Labs, 70435 Stuttgart, Germany.

Xavier Costa-Perez is with the 5G/6G R&D Department, NEC Laboratories Europe, 69115 Heidelberg, Germany, also with the Network Data Science Group, i2CAT Foundation, 08034 Barcelona, Spain, and also with ICREA, 08010 Barcelona, Spain.

Marco Fiore is with the Network Data Science Group, IMDEA Networks Institute, 28918 Madrid, Spain.

Digital Object Identifier 10.1109/TNSM.2025.3580706

cloud services like Amazon AWS or Google Cloud Platform (GCP) consistently exceed 50 seconds for *cold startup* and 25 seconds for *warm startup*, topping at 120 seconds in worst-case scenarios [9]. Cold startup time refers to the VM's startup time when a user creates a new VM, and warm startup time indicates the startup time measured when a user re(initiates) an existing (and stopped) VM.

Even in recent tests, booting a lightweight VM containing an Alpine Linux takes around 30 seconds in a local deployment [10]. Reconfiguring already allocated resources has also a non-negligible cost: modern software architectures such as Kubernetes need several seconds to execute new pods, *e.g.*, on VMs that are already running [10]. In addition, re-orchestration often implies recomputing paths on the transport networks and implementing them via, *e.g.*, Software Defined Networking (SDN) architectures: the latency is in the order of hundreds of milliseconds in a small five-switch topology and with precomputed routing [11], and this figure has to be scaled to thousands of switches with on-the-fly path re-calculation.

All unavoidable delays above entail monetary fees for the operator, in terms of both violations of the SLA with the tenants (*e.g.*, due to infringement of guarantees on end-to-end latency), and user dissatisfaction (with ensuing high churn rates). By neglecting these sources of cost, present capacity forecast solutions risk introducing uncontrolled data flow latency once deployed in operational networks, ultimately causing economic losses to the operator.

Key contributions. In this paper, we propose an original model for the anticipatory allocation of capacity to network slices, which is mindful of all operating costs associated to

- (*i*) unnecessary resource overprovisioning,
- (*ii*) non-serviced demands,
- (*iii*) resource instantiation, and
- (*iv*) resource reconfiguration.

To this end, we adopt a novel approach based on the concept of *multi-timescale orchestration* illustrated in Figure 1.

On the left, a state-of-the-art solution for capacity forecasting [12] tries to accommodate the demand and to limit overprovisioning by reconfiguring resources at every re-orchestration opportunity (top); by doing so, it minimizes costs (*i*) and (*ii*) above (second plot). However, it also ceaselessly instantiates or de-commissions capacity, and reallocates available resources in a sustained way. This incurs in substantial instantiation and reconfiguration fees (third plot) that ultimately lead to a high overall economic cost (bottom).

On the right, our model performs the orchestration at two timescales and by telling apart two classes of resources. A *long-timescale orchestrator* operates over extended intervals that span multiple re-orchestration opportunities; it allocates a *dedicated capacity* to each slice and also reserves an additional *shared capacity* accessible by any slice. Both capacities remain constant across the extended interval, limiting the frequency of instantiation and thus cost (*iii*). Only the shared capacity is then reallocated at every re-orchestration opportunity by a *short-timescale orchestrator*, while the configuration of the dedicated capacity is preserved throughout the extended interval, thus reducing cost (*iv*). Both long- and short-timescale orchestrators decide on the amount of (dedicated and shared) resources to be allocated to each slice to also minimize the usual costs (*i*) and (*ii*). This comprehensive strategy results in a 47% reduction of the total cost in the example in Figure 1. Implementing the proposed approach requires designing a two-timescale model, as summarized in the diagram in Figure 2.

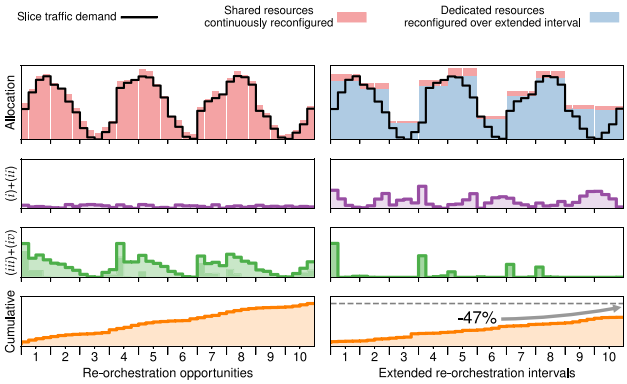

Fig. 1. Toy example illustrating the costs of resource allocation in network slicing. Top: traffic demand generated by a representative slice (black solid line), along with the capacity allocated based on predictions, and reconfigured at all available re-orchestration opportunities (i.e., shared, in red) or only periodically over extended intervals (i.e., dedicated, in blue). Second row: Monetary costs of (*i*) overprovisioning and (*ii*) non-serviced slice traffic. Third row: Monetary costs of resource (*iii*) instantiation and (*iv*) reconfiguration. The costs are obtained with a system configuration $\kappa_O = \kappa_S = \kappa_i = 1$ and $\kappa_r = 0.5$, whose meaning is explained in Section II. Bottom: cumulative overall cost over time, for components (*i*)-(*iv*). Left: a legacy capacity forecasting model [12] updates the prediction at the fastest rate possible, closely following the demand fluctuations but forcing continuous reconfigurations. Right: our proposed multi-timescale capacity forecasting model trades slightly increased overprovisioning for much-reduced instantiation costs (which are only incurred once per extended interval) and reconfiguration fees (which are completely avoided for dedicated resources).

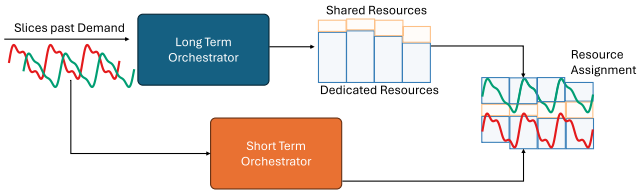

Fig. 2. Overall framework for Long and Short Term Orchestration. The Long Term Orchestrator manages demand slicing based on past usage patterns, while the Short Term Orchestrator handles real-time resource assignment across shared and dedicated resources.

The proposed framework integrates long-term and short-term orchestration mechanisms. The Long Term Orchestrator forecasts and slices demand based on historical trends, whereas the Short Term Orchestrator performs fine-grained resource allocation using available dedicated and shared infrastructure. Interestingly, a multi-timescale orchestration model allows exploring trade-offs such as that associated to the increased overprovisioning occurred when reducing instantiation and reconfiguration costs, as observed in the sample case study in Figure 1. This empowers an unprecedently comprehensive solution for cost-driven orchestration of slice resources [13] via a mixture of AI and traditional optimization.

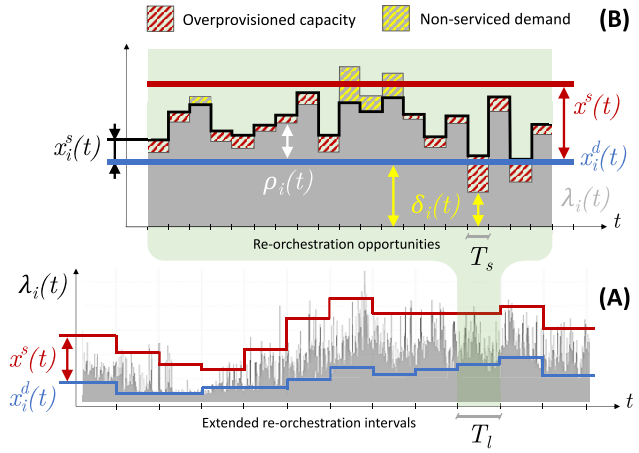

Fig. 3. Orchestration model. (A) Long-timescale orchestration. The background time series represents the traffic demand generated by slice i (grey). The curves portray the time evolution of the dedicated capacity $x_i^d(t)$ allocated to slice i (blue), and of the shared capacity $x^s(t)$ (red) over extended intervals of duration T_l. Note that $x^s(t)$ is added to the dedicated resources to determine the total available capacity, and, unlike $x_i^d(t)$, is not reserved for slice i but available to all slices. (B) Short-timescale orchestration during one extended interval. At every $T_s < T_l$, a portion $x_i^s(t)$ (black solid curve) of the (fixed) shared capacity $x^s(t)$ is allocated to slice i, based on the residual demand $\rho_i(t)$ not satisfied by the (fixed) dedicated resources $x_i^d(t)$. The plot also highlights the volume of overprovisioned capacity and non-serviced demand (pattern regions), and the slice traffic below dedicated capacity $\delta_i(t)$.

II. Orchestration Model and Trade-Offs

Our orchestration model is outlined in Figure 3, which also serves the purpose of illustrating the notation used in the remainder of the paper. Let us denote by $\lambda_i(t)$ the traffic demand generated by services running in slice $i \in \mathbb{S}$ at time t. The long-timescale orchestrator operates on extended intervals of duration T_l. At the beginning of each such interval, it takes decisions on the dedicated capacity $x_i^d(t)$ allotted to slice i, $\forall i \in \mathbb{S}$, and on the additional shared capacity $x^s(t)$ available to all slices; all capacities are conserved throughout the following T_l. The bottom plot (A) in Figure 3 depicts an example of allocation resulting from a long-timescale orchestration.

Within an extended interval, the short-timescale orchestrator assigns resources to each slice i at all re-orchestration opportunities, occurring at every T_s. Decisions are based on the (estimated) future residual demand $\rho_i(t) = \max\{0, \lambda_i(t) - x_i^d(t)\}$, and lead to the allocation of an additional capacity $x_i^s(t)$, for each slice i. The resources $x_i^s(t)$ may be reconfigured at every T_s, and are provisioned on top of the dedicated $x_i^d(t)$. The top plot (B) in Figure 3 illustrates these definitions for a sample short-timescale orchestration during one extended interval.

A. Sources of Monetary Cost

Building on the notation above, we can formally introduce the different costs associated to the management of resources in sliced networks. As anticipated in Section I, there are four sources of economic penalty for the operator, as follows.

***(i)* Unnecessary resource provisioning**: the operator incurs a monetary cost in terms of both Capital Expenditure (CAPEX) and Operating Expenses (OPEX) that is directly proportional to the amount of unused resources it allocates to a slice. Such capacity it is instantiated and configured to no purpose and could be allotted, *e.g.*, to other slices to increase the global system efficiency. This cost at time t is

$$\sum_{i\in\mathbb{S}} f_1\Big(\max\{0, x_i^d(t) - \delta_i(t)\}\Big) + \sum_{i\in\mathbb{S}} f_1(\max\{0, x_i^s(t) - \rho_i(t)\}) + f_1\big(x^s(t) - \sum_{i\in\mathbb{S}} x_i^s(t)\big), \quad (1)$$

where $\delta_i(t) = \min\{\lambda_i(t), x_i^d(t)\}$ denotes the portion of the demand of slice i served by the dedicated capacity at time t, as shown in plot (B) of Figure 3. The first two terms in (1) denote the cost of overprovisioning at slice i and time t, due to the unneeded allocation of dedicated and shared capacity, respectively; again, we refer the reader to plot (B) of Figure 3 for an exemplification. The third term captures instead the overprovisioned shared capacity that is not allocated to any slice by the short-term orchestrator. The function $f_1(\cdot)$ describes the scaling of cost with capacity overprovisioning. As in [12], in our evaluation we assume a linear increase of the penalty, *i.e.*, $f_1(x) = \kappa_o x$, where κ_o is the monetary cost of one unit of capacity and is expressed in \$/Mbps. However, our model can easily accommodate different definitions of the scaling law, which may apply to specific network functions.

***(ii)* Non-serviced demand**: every time the operator does not allocate sufficient resources to serve the traffic demand of a slice, it violates the SLA with the tenant, which triggers a monetary compensation. The associated cost at time t is

$$\sum_{i\in\mathbb{S}} \kappa_s \cdot \mathbb{1}_{<\rho_i(t)}(x_i^s(t)), \quad (2)$$

where $\mathbb{1}_A(x)$ is an indicator function that takes a value 1 if the argument satisfies condition A, and 0 otherwise. Thus, $\mathbb{1}_{<\rho_i(t)}(x_i^s(t))$ activates when the portion of shared capacity assigned to i does not suffice to meet the service demand; this corresponds to an underprovisioning situation, as depicted in Figure 3. In these cases, the operator has to indemnify the tenant for a value κ_s, in \$, per SLA violation. This definition is also in line with those used in the literature [12].

***(iii)* Resource instantiation**: in presence of substantial variations of the total traffic demand, the operator needs to instantiate new resources to serve the demand of the slice. In these cases, as discussed in Section I, there exists a cost associated to enabling such new resources. As an example, if additional Virtual Machines (VMs) have to be bootstrapped or migrated to serve the slice, the operator has increased expenses in terms of power consumption and CPU cycles. In addition, there may be an indirect penalty in terms of perceived Quality of Service (QoS), as this operation may take minutes [10] and

disrupt the end-user experience. The cost, triggered at every T_l in our multi-timescale model, can be modeled as

$$\sum_{i\in\mathbb{S}} f_2(\delta_i(t)) \cdot \mathbb{1}_{>x_i^d(t-1)}\left(x_i^d(t)\right) + f_2(\min\{\rho_i(t), x_i^s(t)\}) \cdot \mathbb{1}_{>x^s(t-1)}(x^s(t)). \quad (3)$$

The first term in (3) represents the penalty incurred when new dedicated resources must be instantiated, which occurs when $x_i^d(t) > x_i^d(t-1)$. The second term is equivalent to the first one, but refers to the shared capacity instantiated to all slices in $\mathbb{S}$. Note that the costs induced by both terms are functions $f_2(\cdot)$ of the affected traffic that may experience disruption, *i.e.*, $\delta_i(t)$ and $\min\{\rho_i(t), x_i^s(t)\}$, respectively.[1] In our performance evaluation, we consider the cost to be directly proportional to the affected traffic, *i.e.*, $f_2(x) = \kappa_i x$, where the parameter κ_i captures the estimated fee for delaying one unit of capacity due to resource instantiation, expressed in \$/Mbps.

***(iv)* Resource reconfiguration**: while resources are only instantiated at every T_l, a short-timescale orchestration of the shared capacity within each extended interval allows accommodating faster fluctuations of the slice demand. Every time the operator reconfigures the shared capacity, it incurs a cost; as mentioned in Section I, this is the case with the reconfiguration of the SDN transport networks, the setup of load balancers, or the creation of new instances of a VNF on a VM previously used by another slice. All these operations have a price in terms of management delay [10], expressed as

$$\sum_{i\in\mathbb{S}} f_3(\min\{\rho_i(t), x_i^s(t)\}) \cdot \mathbb{1}_{\neq x_i^s(t-1)}(x_i^s(t)). \quad (4)$$

The above cost is present whenever the shared resources must be reconfigured for a slice *i*, *i.e.*, $x_i^s(t) \neq x_i^s(t-1)$. In such situations, the cost is dependent on the amount of traffic affected by the reconfiguration process, *i.e.*, $\rho_i(t)$ bounded by $x_i^s(t)$. In our study, we assume that the economic penalty is the same for any bit of traffic using reconfigured resources, hence $f_3(x) = \kappa_r x$, where κ_r is in \$/Mbps. Also, in this case, other functions can be easily embedded in the overall framework to represent distinctive cost models identified by the operator.

B. Trade-Offs in Capacity Allocation

The basic trade-off in anticipatory resource assignment is that between overprovisioning and non-serviced demands.

- *Trade-off A.* Increasing the amount of resources makes overprovisioning more likely, but reduces the probability that the allocated capacity is not sufficient to serve the future demand. This results in opposing costs (*i*) and (*ii*).

Current capacity forecasting models aim at identifying the optimal compromise that minimizes the joint penalty of the costs in trade-off A above [12]. However, these models do not offer any control over instantiation and reconfiguration. By adopting a multi-timescale approach, we are instead capable of factoring such variables in. Specifically, the model presented in Section II-A tells apart the capacity allocated to each slice into a dedicated capacity, re-orchestrated over long timescales with period T_l, and a shared capacity, re-orchestrated over short timescales with period T_s. This unlocks additional degrees of freedom: the orchestrator can decide not only how many resources to assign to a slice, but also which portion of those shall be of each type, and for how long they stay unaltered.

Our model still allows addressing trade-off A above, by controlling the total allocated capacity during each extended interval T_l, *i.e.*, $x^s(t) + \sum_{i\in\mathbb{S}} x_i^d(t)$, and modulate the costs of overprovisioning and non-serviced demands. Yet, its flexibility enables the exploration of the following additional trade-offs.

- *Trade-off B.* By increasing the dedicated capacity $x_i^d(t)$ allocated to slice *i* during an extended interval, the orchestrator can serve a larger fraction of the slice traffic with resources that do not need reconfiguration. However, such resources cannot be reused by other slices during T_l whenever they are not needed by slice *i*. For instance, in plot (B) of Figure 3, increasing $x_i^d(t)$ would reduce the residual demand $\rho_i(t)$ that is served with reconfiguration-heavy shared capacity; but it would also generate additional overprovisioning, *e.g.*, in the fourth and second to last re-orchestration opportunities. This leads to a trade-off between costs (*i*) and (*iv*).
- *Trade-off C.* Allocating a larger shared capacity $x_i^s(t)$ to slice *i* during an interval T_s reduces the risk that the resources will not be sufficient to serve the future slice demand. Nevertheless, it also causes the reconfiguration of more resources. As an example, in plot (B) of Figure 3, increasing $x_i^s(t)$ in the third T_s slot could remove the non-serviced demand, but would also grow the reconfiguration penalty. A trade-off exists between costs (*ii*) and (*iv*).
- *Trade-off D.* Increasing the duration T_l of the extended interval reduces the cost of resource instantiation, which only occurs once per extended interval. However, a higher T_l also forces the dedicated capacities $x_i^d(t)$ and the total shared capacity $x^s(t)$ to remain constant for a longer time. With a reduced capability to tailor the network resources to the fluctuations of the slice traffic demands, the orchestrator may incur in increased overprovisioning or underprovisioning. For instance, extending the timespan of plot (B) in Figure 3 to cover a prolonged demand $\lambda_i(t)$ may create additional situations where $x_i^d(t) > \lambda_i(t)$, *i.e.*, dedicated resources go wasted, as in the second to last T_s interval; it can also generate new cases where $x_i^s(t) < \rho_i(t)$, and traffic peaks are not serviced, as in the central part of the example. This results in a trade-off between cost (*iii*) and joint costs (*i*) and (*ii*).

Next, we present a framework that takes automated, anticipatory decisions on capacity allocation by addressing all trade-offs outlined above, thanks to the multi-timescale model.

[1] Instantiation and reconfiguration costs are proportional to the full amount of impacted demand (rather than, *e.g.*, to the increment of demand with respect to the previous re-orchestration opportunity) since finding a stable and optimal allocation of resources typically requires reconsidering their organization within a large portion of the network, or even across all slices [14].

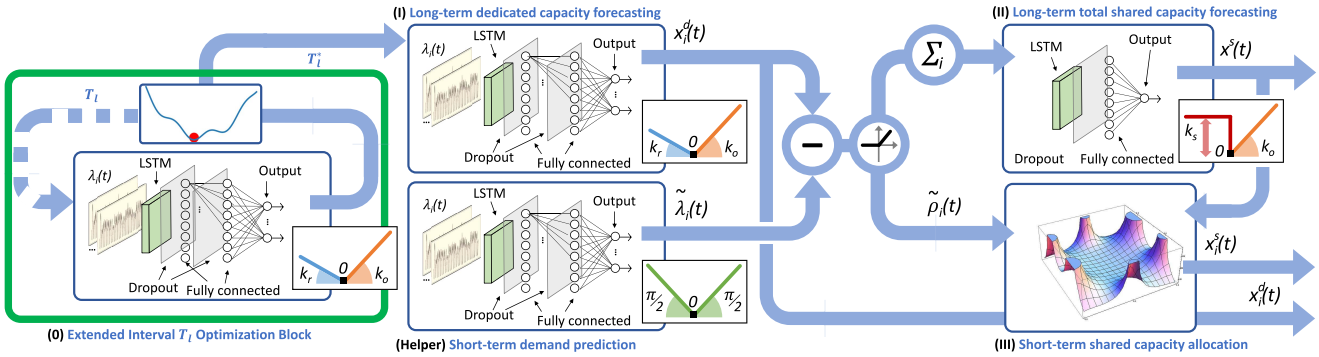

Fig. 4. Overview of the AZTEC+ framework. The learning flow proceeds from left to right. The input mobile data is processed by deep neural networks that, jointly with a Golden-Section Search algorithm, find the optimal Extended interval T_l (Block 0), which is the addition for AZTEC+, in the green edges. Then, for each slice $i \in S$, a deep neural network returns the long-term dedicated capacity $x_d^i(t)$, having as input not only the mobile data, but also the optimal extended interval T_l^*, and the short-term estimated demand $\lambda_i(t)$, respectively. These values are combined to obtain the estimated residual demands $\hat{\rho}_i(t)$. The aggregate residual demand over all slices is input to a further deep neural network to determine the long-term shared capacity $x_s(t)$. Such capacity is then fed, along with per-slice residuals, to an optimization module that allocates the shared resources $x_s^i(t)$.

III. The AZTEC+ Framework

Our framework, named AZTEC+ (*i.e.*, capacity Allocation for Zero-Touch nEtwork sliCing), automatically solves trade-offs A, B, and C above by finding an effective compromise among the opposing goals of reducing the operator's costs in terms of (*i*) overprovisioning, (*ii*) non-serviced slice demands, and (*iv*) resource reconfiguration. In addition, AZTEC+ dynamically and automatically selects the extended interval T_l^* that minimizes the penalty associated with (*iii*) capacity instantiation, to address trade-off D. Next, we first provide an overview of the framework, and then discuss the implementation of its different components for long- and short-timescale orchestration and for extended interval selection.

Block (0) selects the Extended Interval T_l^* that minimizes the instantiation and reconfiguration costs through the Golden-Section Search Algorithm [15]. To do so, it performs the forecasting of the long-term dedicated capacity for each network slice $x_d^i(t)$, using as input information, the actual traffic generated by each slice during the preceding N re-orchestration opportunities and a candidate extended interval T_l. During one cycle of the Golden-Section Search Algorithm, the Deep Neural Network (DNN) is trained using the candidate T_l value, and its validation loss is evaluated. The Golden-Section Search Algorithm, progressively adjusting T_l, converges to the T_l^* that returns the minimum validation loss. The validation loss is computed as the sum of the reconfiguration and instantiation costs. The cycle is repeated 10 times, and the Extended Interval T_l, which achieves the lowest validation loss, is selected.

Block (I) performs the forecasting of the long-term dedicated capacity for each network slice $x_i^d(t)$, using as input information about the actual traffic generated by each slice during the preceding N re-orchestration opportunities. As discussed for trade-off B in Section II-B, the capacity $x_i^d(t)$ modulates the impact of (*i*) provisioning unnecessary dedicated resources versus (*iv*) re-configuring the shared resources needed to serve the residual demand beyond $x_i^d(t)$. Hence, block (I) identifies $x_i^d(t)$ minimizing costs (*i*) and (*iv*). To this end, block (I) employs the following loss function:

$$LossBlock_1(x) = \begin{cases} \kappa_r \cdot x & \text{if } x \leq 0 \\ \kappa_o \cdot x & \text{if } x > 0. \end{cases} \quad (5)$$

Block (II) determines the long-term shared capacity $x^s(t)$ available to any slice during the subsequent time interval T_l. The shared capacity is used to serve the residual demands of all slices; hence $x^s(t)$ shall be dimensioned to the aggregate residual traffic $\sum_{i\in\mathbb{S}} \rho_i(t)$. Thus, block (II) receives as input an estimate of such aggregate during the previous extended interval, *i.e.*, $\sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)$, $t \in [t - (T_l/T_s), t-1]$, and predicts the next $x^s(t)$ such that (*i*) overprovisioning of shared resources is reduced as much as possible, and (*ii*) all residual demands can be accommodated within $x^s(t)$. In this way, block (II) addresses trade-off A, jointly minimizing costs (*i*) and (*ii*). To this end, block (II) employs the following loss function inspired by α-OMC [16]:

$$LossBlock_2(x) = \begin{cases} \kappa_s - \epsilon \cdot x & \text{if } x \leq 0 \\ \kappa_s - \frac{1}{\epsilon} x & \text{if } 0 < x \leq \epsilon\kappa_s \\ \kappa_o \cdot x - \epsilon\kappa_s & \text{if } x > \epsilon\kappa_s. \end{cases} \quad (6)$$

Each approximate $\tilde{\rho}_i(t) = \max\{0, \tilde{\lambda}_i(t) - x_i^d(t)\}$ is computed from a forecast $\tilde{\lambda}_i(t)$ returned by a *helper* legacy traffic predictor that forecasts per-slice demands over the next re-orchestration opportunity t, as per Figure 4. Note that, at this stage, the actual residuals $\rho_i(t)$ could be directly determined from the (known) real traffic demand $\lambda_i(t)$ observed during the previous interval T_l. However, we opt to feed block (II) with an estimate based on $\tilde{\lambda}_i(t)$ to ensure overall system consistency. Indeed, the short-timescale orchestrator – implemented by block (III) and presented next– necessarily operates on the predicted residuals $\tilde{\rho}_i(t)$ over the future T_s interval. Considering the same estimates in the long-timescale module allows allocating the shared capacity $x^s(t)$ in a way that is conscious of the inaccuracy of the information available during the following short-term resource assignment phase of the framework.

Once the long-term capacities $x_i^d(t)$, $\forall i \in \mathbb{S}$, and $x^s(t)$ are set, the short-timescale orchestrator assigns portions $x_i^s(t)$ of

the total shared resources to each slice. This allocation occurs at every re-orchestration opportunity, spaced by T_s, and is carried out by block (III) of the AZTEC+ framework as follows.

Block (III) receives as input the long-term shared capacity $x^s(t)$, and the future residual demand $\tilde{\rho}_i(t)$ expected for each slice i during the following T_s. The available total capacity is allotted to each slice in a way to solve the trade-off C described in Section II-B. Therefore, block (III) computes $x_i^s(t)$ for each $i \in \mathbb{S}$, by minimizing the combination of costs (*ii*) and (*iv*), corresponding to insufficient allocated capacity and additional shared resource reconfiguration, respectively.

Overall, blocks (I)-(III) return a forecast of all capacities $x_i^d(t)$, $x^s(t)$ and $x_i^s(t)$ that the operator shall allocate over both long and short intervals of duration T_l^* set by block (0), and T_s, respectively. The resulting anticipatory allotment reduces the network management costs associated with the provisioning of exceeding or inadequate resources and their reconfiguration over time.

We remark that differently from AZTEC, AZTEC+ takes automated decisions on the value of the extended re-orchestration interval, T_l in order to tackle the penalty of network resource instantiation. As explained above, control on instantiation costs is achieved by block (0) that automatically and dynamically sets T_l to cope with trade-off D. Thanks to the addition of block (0), the system is not only more reliable due to the automatic selection of T_l (avoiding the human in the loop), but also more adaptive to other scenarios.

Having clarified the role of each block, we detail in the following their implementation, which leverages a combination of deep learning and numerical optimization methods.

IV. Long-Timescale Resource Assignment

The long-timescale orchestration is carried out by blocks (0), (I) and (II) of the AZTEC+ framework, as follows:

A. Forecasting for Extended Interval and Dedicated Capacity

The DNN architecture of Block (0) and Block (I) is illustrated in the respective components of Figure 4. It consists of an Long-short Term Memory (LSTM) layer with 128 cell units and two Fully Connected (FC) layers of 64 and $\|\mathbb{S}\|$ neurons, respectively, where the operator $\|\cdot\|$ returns the cardinality of the argument set. The layers are interleaved by two dropout layers [17].

All layers employ Rectified Linear Unit (reLU) as the neuron activation function, except for a linear function in the last FC layer. The loss function that drives the DNN training is a custom expression designed to account for the actual management costs incurred by the operator in case of errors in the orchestration of the dedicated capacity. If the operator were able to allocate to a slice i a constant capacity $x_i^d(t)$ that perfectly matched the actual demand $\lambda_i(t)$ over the next T_l, the error and cost would be nil: this is the ideal scenario where all the demand is serviced, without any overprovisioning or re-configuration. However, in practical cases, it is impossible to perfectly predict $\lambda_i(t)$, which is also very unlikely to be constant over the whole T_l. In this case, positive errors $x_i^d(t) - \lambda_i(t)$ lead to overprovisioning, with a cost set by the first term of (1) in Section II-A, and negative errors imply that the demand in excess of $x_i^d(t)$ needs to be served by the shared capacity, with (4) setting the re-configuration penalty.[2]

Positive errors yield $\delta_i(t) = \lambda_i(t)$, while $\rho_i(t) = \lambda_i(t) - x_i^d(t)$ for negative errors. Then, the loss function for $x_i^d(t)$ allocated at t is $\sum_{t\in\mathbb{T}} \ell_i^{(\mathrm{I})}(t)$, where $\mathbb{T}$ is the set of concerned re-orchestration opportunities $\{t, \ldots, t + (T_l/T_s) - 1\}$, and

$$\ell_i^{(\mathrm{I})}(t) = \begin{cases} f_3\Big(\lambda_i(t) - x_i^d(t)\Big) & \text{if } x_i^d(t) \le \lambda_i(t) \\ f_1\Big(x_i^d(t) - \lambda_i(t)\Big) & \text{otherwise}. \end{cases} \tag{7}$$

Importantly, (7) has partial derivatives that form a piece-wise constant function, which guarantees robust and fast convergence under popular first-order optimizers like Adam [18].

In order to further improve the quality of the allocation of dedicated resources, we leverage a recent result in neural network design, which allows estimating the uncertainty of the learning outcome. Specifically, adding dropout layers during model testing is mathematically equivalent to generating an approximation of the probabilistic deep Gaussian process [19]. This observation, which holds for DNNs with arbitrary depth and non-linearities, provides a way to return a distribution instead of a scalar output value. We thus activate the dropout layers during testing, and adopt a Monte Carlo strategy; namely, we perform the forward pass L times for each test input, obtaining outputs $\{x_i^{d,1}(t), \ldots, x_i^{d,L}(t)\}$ for, slice i at time t. We then compute the mean $\mu_i^d(t) = 1/L \cdot \sum_{l=1}^{L} x_i^{d,l}(t)$ as well as the variance $\sigma_i^d(t) = 1/L \cdot \sum_{l=1}^{L} (\mu_i^d(t) - x_i^{d,l}(t))^2$, and approximate the model uncertainty as $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$.

Knowledge of the model uncertainty allows adding a safety margin to the standard DNN outcome. Since the whole support of $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$ represents potentially correct values of the dedicated capacity, block (I) returns the 99$^{\text{th}}$ percentile of the distribution; this makes it very unlikely to output a $x_i^d(t)$ that is lower than the best one, minimizing the risk of SLA violations. Thus, when the DNN is confident about the quality of the result, it returns a value close to the mean; otherwise, it adds a substantial safety margin. An example is in Section VI-A.

B. Forecasting for Long-Term Shared Capacity

Block (II) is implemented using a dedicated DNN, although with a simpler structure due to the reduced richness of the input. The DNN is fed with a single time series of the total residual demand in the past extended interval, *i.e.*, $\sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)$, $t \in [t - (T_l/T_s), t - 1]$. The input is processed by three FC layers with 128, 64 and 1 neurons; the first two use reLU activation functions, while the last uses a linear function. A dropout layer is between the first and second FC layers.

The DNN is trained with a different custom loss function that accounts for the correct sources of monetary penalty in case of errors. An ideal case where a fixed long-term shared capacity $x^s(t)$ perfectly matches a constant aggregate residual

[2] As the long-term orchestrator is agnostic of the short-timescale operation, it cannot factor $x_i^s(t)$ in the cost of negative errors. So, block (I) assumes perfect management of the shared capacity that always accommodates a continuously varying residual demand, reducing (4) to $\sum_{i\in\mathbb{S}} f_3(\rho_i(t))$.

demand is unrealistic, and errors are unavoidable. Positive errors yield overprovisioning of $x^s(t)$, whereas negative ones have a cost in terms of denied traffic. The former corresponds to the second and third terms[3] in (1), while the latter maps to the monetary fee[4] for SLA violations in (2). The loss function jointly capturing these costs for the extended interval starting at t is $\sum_{t\in\mathbb{T}} \ell_i^{(\mathrm{II})}(t)$, where

$$\ell_i^{(\mathrm{II})}(t) = \begin{cases} \kappa_s & \text{if } x^s(t) < \sum_{i\in\mathbb{S}} \tilde{\rho}_i(t) \\ f_1\big(x^s(t) - \sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)\big) & \text{otherwise}. \end{cases} \tag{8}$$

The expression in (8) needs to be slightly modified by adding minimum slopes that make the function differentiable over $\mathbb{R}$. With this, the loss function has the same desirable properties as the ones mentioned for (7). Finally, we take advantage of the dropout layer also in this case: we thus approximate the model uncertainty, and return the 99$^{\text{th}}$ percentile of the distribution as a safety margin on the correct value of $x^s(t)$.

V. Short-Timescale Resource Assignment

The short-timescale orchestration consists of block (III) supported by a *helper* short-term demand predictor, as outlined in Figure 4. The predictor uses a DNN architecture that is very similar to that adopted by block (I); indeed, the two DNNs operate on the same input, and produce per-slice forecasts. The main differences between them are in the frequency of operation and, most notably, in the loss function. The helper predictor outputs a prediction at every T_s instead of at every T_l. Furthermore, it uses a traditional Mean Absolute Error (MAE) instead of the cost-aware loss function in (7): MAE considers identical contributions by the positive and negative errors, thus producing an output $\tilde{\lambda}_i(t)$ that tries to follow as closely as possible the upcoming slice traffic demands.

Given the total shared capacity $x^s(t)$, and the estimated residual demands $\tilde{\rho}_i(t)$, AZTEC has to decide how to distribute $x^s(t)$ across the requesting slices at every T_s. This is implemented in block (III) using numerical optimization.

The primary objective of the shared resource assignment performed by block (III) is avoiding SLA violations due to insufficient available capacity, which would induce a cost modeled in (2). At the same time, issuing non-essential resources has a penalty in terms of unnecessary reconfiguration cost, as captured by the expression in (4). This corresponds to trade-off C in Section II-B, and can be formulated as:

$$\begin{aligned} \min_{x_i^s(t)} \quad & \sum_{i\in\mathbb{S}} \kappa_s P(\tilde{\rho}_i(t) > x_i^s(t)) \\ \text{subject to} \quad & \sum_{i\in\mathbb{S}} x_i^s(t) \le x^s(t), \end{aligned} \tag{9}$$

[3]Also in this case, the shared capacity allotted to individual slices $x_i^s(t)$ used in (1) has not yet been determined at this stage. The safest option is hence to assume that the whole residual demands will be correctly assigned by the short-term orchestrator. This leads to approximating the allocated shared resources $x_i^s(t)$ by $\tilde{\rho}_i(t)$. Under this hypothesis, the second and third terms in (1) reduce to 0 and $f_1(x^s(t) - \sum_{i\in\mathbb{S}} \tilde{\rho}_i(t))$, respectively.

[4]By assuming $x_i^s(t) = \tilde{\rho}_i(t)$, unserviced demands are possible only when $x^s(t)$ is insufficient. Then, (2) translates into $\kappa_s \cdot \mathbb{1}_{<\sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)}(x^s(t))$.

Algorithm 1: Shared Resource Assignment Algorithm

Function *TRANSFORM*$(p_1, \ldots, p_{\|\mathbb{S}\|})$**:**
  $x^s_{\|\mathbb{S}\|} = \frac{p_{\|\mathbb{S}\|}\, x^s}{\left(\sum_{i=1}^{\|\mathbb{S}\|-1} \prod_{j=i}^{\|\mathbb{S}\|-1} \frac{p_j}{1-p_j}\right)+1}$
  $x_i^s = \left(\prod_{j=i}^{\|\mathbb{S}\|-1} \frac{p_j}{1-p_j}\right) x^s_{\|\mathbb{S}\|},\ i \in [1, \|\mathbb{S}\| - 1]$
  return $x_1^s, \ldots, x^s_{\|\mathbb{S}\|}$
Function *COST*$(p_1, \ldots, p_{\|\mathbb{S}\|})$**:**
  $x_1^s, \ldots, x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM*$(p_1, \ldots, p_{\|\mathbb{S}\|})$
  $X \leftarrow \sum_i \kappa_s P\big(\tilde{\rho}_i > x_i^s\big)$
  return X
Function *MAIN*$(x^s, \tilde{\rho}_i)$**:**
  $c \leftarrow$ BOBYQA(COST$(p_1 = 0.5, p_2 = 0.5, \ldots, p_{\|\mathbb{S}\|-1} = 0.5, p_{\|\mathbb{S}\|})$)
  $p_1^0, \ldots, p^0_{\|\mathbb{S}\|} \leftarrow$ *GOLDEN* $(c(p_{\|\mathbb{S}\|}))$
  $p_1, \ldots, p_{\|\mathbb{S}\|} \leftarrow$ *BOBYQA* $(p_1^0, \ldots, p^0_{\|\mathbb{S}\|})$
  $x_1^s, \ldots, x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM*$(p_1, \ldots, p_{\|\mathbb{S}\|})$
  return $x_1^s, \ldots, x^s_{\|\mathbb{S}\|}$

The above minimizes the expected value of the expenditure for non-serviced slice demands in (2). Note that, by squeezing as many slices as possible within the total capacity, the solution to (9) implicitly addresses the challenge of minimizing reconfiguration cost. Also, the formulation in (9) deals with probabilities: this is consistent with the probabilistic nature of $\tilde{\rho}_i(t)$ granted by uncertainty approximation via dropout layers.

Due to the empirical nature of the probability distribution $\tilde{\rho}_i(t)$, (9) has no closed form and thus we cannot employ classical optimization methods. Furthermore, as we do not have a differentiable objective function, we cannot apply approaches that depend on gradients. Hence, we resort to numerical methods that search for the optimal solution within the feasible set of $[x_i^s(t)]$. The following change of variable

$$p_i(t) = \begin{cases} x_i^s(t) / \big(x_i^s(t) + x_{i+1}^s(t)\big) & \text{if } i \in [1, \|\mathbb{S}\| - 1] \\ \sum_i x_i^s(t) / x^s(t) & \text{if } i = \mathbb{S} \end{cases} \tag{10}$$

yields $p_i(t) \in [0, 1],\ \forall i \in \mathbb{S}$, which simplifies the search to the N-dimensional variable space with fixed bounds $[0, 1]$ on all variables. Note that the first $\|\mathbb{S}\| - 1$ variables $p_i(t)$ represent the relative amount of shared capacity assigned to slice i with respect to the slice i+1, while the last $p_{\|\mathbb{S}\|}(t)$ represents the overall amount of shared capacity assigned to the services. Therefore, the variable change in (10) serves the following purposes: first, the constraint on the sum of the variables is now enforced through a constraint on each variable; second, we avoid ties between variables, allowing a safe exploration of the solution space where we can focus one variable, and changing its $p_i(t)$ value within the entire range without impacting any of the other variables $p_j(t)$ for $j \neq i$.

Algorithm 1 details the numerical solution for shared resource assignment adopted by AZTEC. A main function takes as input the total available shared capacity $x^s(t)$ and the empirical distribution of the capacity needed by each service

in the next time interval $\tilde{\rho}_i(t)$. Two helper functions, COST and TRANSFORM, compute the cost expected value for a given assignment $p_1(t), \ldots, p_{\|\mathbb{S}\|}(t)$, and transform $p_i(t)$ back to $x_i^s(t)$, respectively. Thanks to the variable change in (10), we can use a gradient-free algorithm that works with constrained input variables for the minimization of (9). In particular, we used the BOBYQA algorithm [20], a gradient-free optimizer that supports constrained variables.

All ratios of shared capacity are initialized to 0.5, *i.e.*, all slice start with the same amount of resources. However, given the possibly high number of slices that can be included in the system, there may be different local minima and there exists the chance of getting stuck in a local minima that does not deliver a good performance. To reduce the probability that this happens, we perform a preliminary search for the best starting $p_{\|\mathbb{S}\|}(t)$, via the Golden Section method [21] and considering $p_{\|\mathbb{S}\|}(t)$ as the only variable for the cost.

VI. Evaluation Results

In order to assess the performance of AZTEC+ for the target task of anticipatory resource provisioning in network slicing, we leverage dependable evaluation scenarios that are based on substantial measurement data. Specifically, we employ real-world traffic generated by a variety of popular mobile services including Google, Netflix and Twitter and assign each service to an independent network slice. The traffic demands are collected in the production network of a major European operator using passive measurement probes deployed in the Evolved Packet Core (EPC). The probes run commercial and proprietary traffic classifiers to identify the service associated to each IP flow. The traffic demands refer to the data generated by mobile subscribers in several major metropolitan areas in Europe, each encompassing a variety of urban, suburban and rural territories. The data consist of the traffic loads served by each of the hundreds of base station covering the target geographical region, at a time granularity of 6 minutes during 11 weeks.

A. Harnessing the Forecast Uncertainty

As presented in Section III, we leverage a recent result on the approximation of uncertainty in DNN to include a safety margin in the model forecast. As a preliminary step in our evaluation of AZTEC+, we investigate the impact of including the knowledge of the estimated uncertainty in the forecast produced by the different DNNs that are part of the framework.

Figure 5 visually shows the benefit of this design choice, by juxtaposing the performance of AZTEC+ with and without uncertainty; in the second case, dropout layers are deactivated during test, and all DNNs produce a single-value output. In each plot, we report the anticipatory allocation of capacities $x_i^d(t)$ and $x_i^s(t)$ to the target slice *i*, as well as that of the total shared capacity $x^s(t)$; the actual demand is on the background.

The plots illustrate well how the time-varying resource allocation achieved by AZTEC+ nicely follows the fluctuations of the slice traffic. More interestingly, AZTEC+ (in the top plot) achieves a more reliable assignment of slice resources by accounting for the level of uncertainty of the predictions.

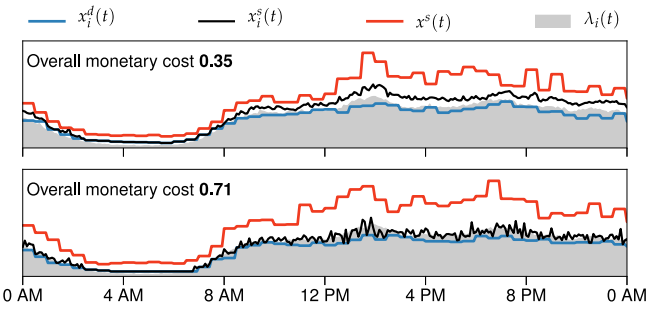

Fig. 5. Time series of sample resource allocations. Top: AZTEC+ framework. Bottom: equivalent framework where no uncertainty estimates are used.

The total capacity $x_i^d(t) + x_i^s(t)$ is smoother and avoids situations where the demand cannot be serviced. Conversely, the framework not accounting for uncertainties (in the bottom plot) yields a capacity allocation that is noisy and incurs in substantial SLA violations due to unsatisfied demands.

In addition, AZTEC+ achieves such a result while saving on the amount of allocated resources (note the lower $x^s(t)$ curve), which ultimately results in an overall monetary cost that is half of that incurred by the framework without uncertainties. While this is an excerpt from a specific test, we recorded similar gains for all settings explored in our analysis.

B. Capacity Breakdown

Fig. 6 illustrates the capacity breakdown for different instantiation penalties (i.e., $k_i = 1$ (Fig. 6(a), $k_i = 10$ (Fig. 6(b), $k_i = 50$ (Fig. 6(c), $k_i = 100$ (Fig. 6(d)) as well as reconfiguration penalties (i.e., $k_r = 0.1$, $k_r = 0.5$, $k_r = 1$, $k_r = 10$)). Across all results, a consistent trend is observed: a greater proportion of traffic is assigned to the more flexible shared capacity when the reconfiguration fee is low, while redirecting traffic to the dedicated capacity becomes more cost-effective as k_r increases. Indeed, over-provisioning dedicated resources becomes more practical than reconfiguring shared resources as k_r grows. Additionally, we can also notice that the usage of shared capacity is higher as k_i increases.

AZTEC+ tends to have higher SLA violation when using more resources from the shared capacity (low reconfiguration penalty k_r) than when using more resources from the dedicated capacity. This is a direct consequence of the selection of longer T_l^*. We observe a high variation in the usage and distribution of shared/dedicated capacity depending on the instantiation penalty cost, ranging from 70% of dedicated capacity when reconfiguration penalty $k_r = 1$ and instantiation penalty $k_i = 1$ to a 30% when $k_r = 0.1$ and $k_i = 100$. Indeed, when the reconfiguration cost is higher, the system tends to allocate the maximum dedicated capacity for each slice. Finally, there is a clear trade-off between k_r and k_i and their impact on resource allocation. For higher instantiation penalties and lower reconfiguration penalties, more resources are assigned to shared capacity x_s. Conversely, for higher reconfiguration penalties, resources are allocated to dedicated capacity x_d.

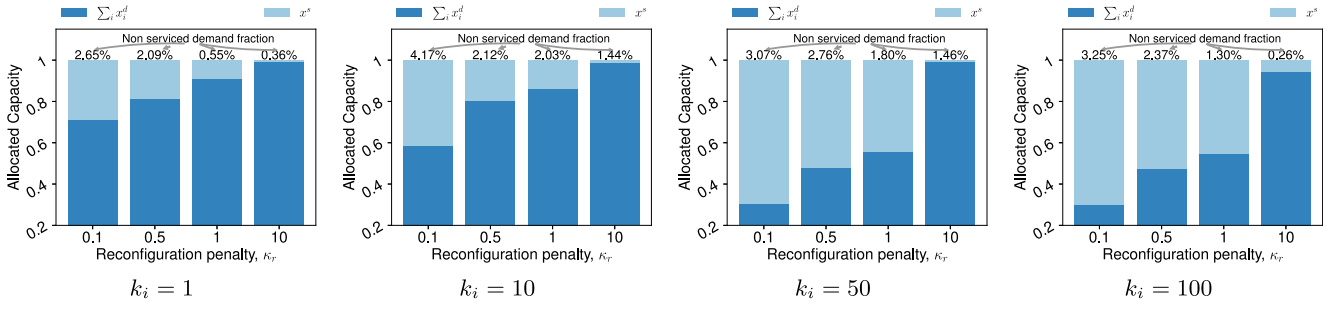

Fig. 6. Total dedicated capacity $\sum_{i\in\mathbb{S}} x_i^d(t)$ and shared capacity $x^s(t)$ allocated by AZTEC+ versus κ_r. Numbers denote the fraction of re-orchestration opportunities with insufficient allocated resources.

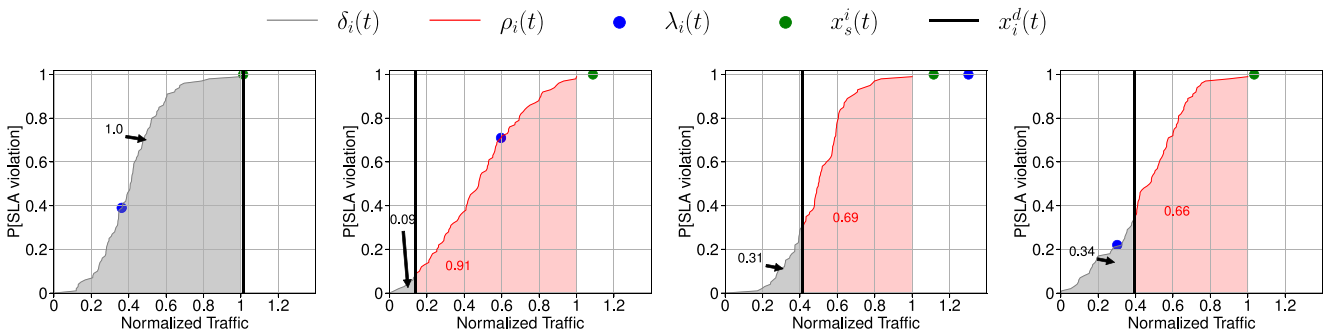

Fig. 7. Different scenarios of cumulative probability of SLA violation. (a) No allocation of shared resources. (b) SLA minimization. (c) Maximum allocation of shared resources. (d) Overprovisioned allocation.

C. Block III Internals

In Figure 7, the detailed operativity of Block (III) is illustrated. The vertical black line represents the actual value of the dedicated capacity $x_d^i(t)$. The grey line represents the probability that the real load is below the dedicated capacity $(\delta_i(t))$ and the red line represents the probability that the real load exceeds the dedicated capacity $(\rho_i(t))$, potentially leading to an SLA violation. The probability distribution is obtained from the Helper output. The blue dot indicates the actual real traffic value $\lambda_i(t)$, which Block (III) is unaware of. Block (III), by solving the optimization problem, assigns shared capacity to slice *i*, aiming at minimizing the red area. The green dot represents the assigned shared capacity for slice *i*, $x_s^i(t)$, derived from the Block (III) output. There are four possible scenarios to consider, as follows.

(*i*) *No allocation.* If given the dedicated capacity allocated to slice *i* by Block (I), the probability of an SLA violation is 0, then Block (III) does not allocate resources from the shared capacity (see Fig. 7(a). Here, Block (III) observes the probability distribution derived from the helper output and the value of static capacity $x_d^i(t)$ allocated by Block (I) and decides not to allocate resources from $x_s(t)$.

(*ii*) *SLA minimization.* If given a certain dedicated capacity $x_d^i(t)$, the probability of an SLA violation is high, then Block (III) allocates resources from the shared capacity (see Fig. 7(b). A high probability of incurring an SLA violation is measured, and then enough resources from $x_s(t)$ are allocated to slice *i* to minimize the probability of incurring SLA violations.

(*iii*) *Max. allocation.* The optimization problem solved by Block (III) allocates shared capacity to slice *i* to minimize the probability of SLA violations. In a certain scenario, the allocated shared capacity is enough to minimize the probability of SLA violations based on the information obtained from Helper, but the actual real traffic value $(\lambda_i(t))$ exceeds the allocated resources $x_i^s(t)$. This situation could occur for two reasons: either the forecast provided by Helper for that timeslot was not accurate enough, or there were not enough shared capacity resources available. In the latter case, Block (III) aims to allocate as many resources as possible to minimize the SLA violation penalty.

(*iv*) *Overprovisioned allocation.* A last scenario may happen when Block III allocates resources from the shared capacity $x_s(t)$ that are not currently needed. In this scenario, the information provided by the Helper suggests that there is a high probability of violating the SLA by only allocating the dedicated capacity $x_i^d(t)$. So, Block (III) allocates shared resources to slice *i* with the intent of preventing SLA violations. However, upon computing the actual value $\lambda_i(t)$, the shared resources allocated were not necessarily needed as the real traffic was already below the dedicated capacity allocated by Block (I). As a result, there is an unnecessary overprovisioning of resources (see Fig. 7(d). It is important to note that Block (III) operates only leveraging information provided by the output of Helper, Block (I), and Block (II), and not using the actual real value $\lambda_i(t)$ that is not known at the time of allocating resources.

D. Instantiation Cost Analysis

Fig. 8 shows the contribution of only the instantiation to the overall cost for all the different configurations (k_r 0.1-10, k_i 1-100). We observe that for low values of reconfiguration

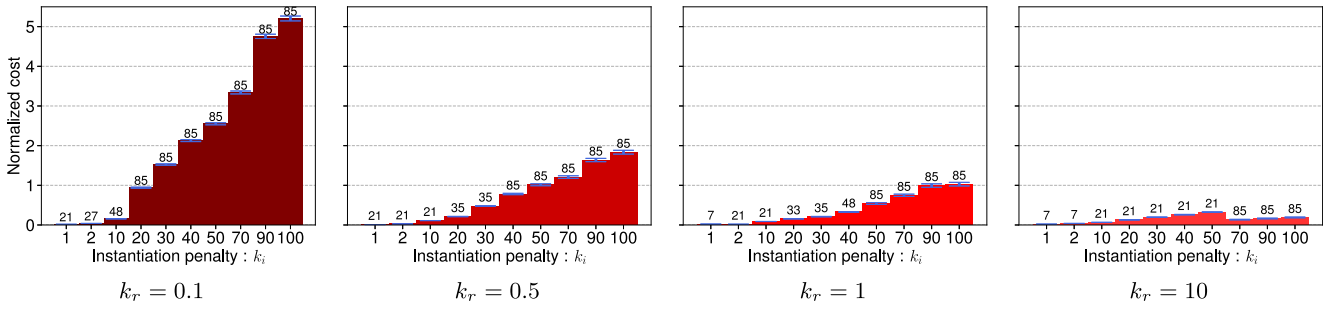

Fig. 8. Normalized instantiation cost for reconfiguration penalties $k_r = 0.1$ (a),$k_r = 0.5$ (b),$k_r = 0.1$ (c), $k_r = 10$ (d).

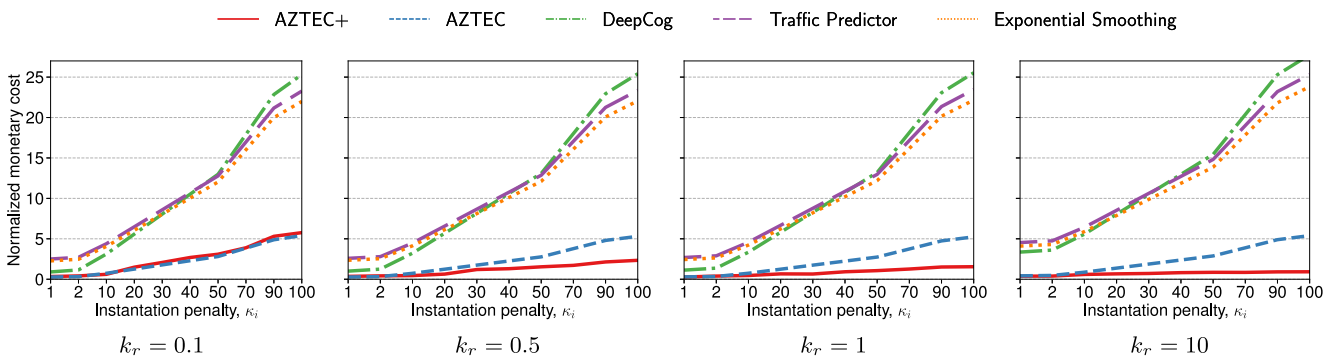

Fig. 9. Normalized monetary cost of AZTEC+ and benchmarks as a function of instantiation penalties k_i for different reconfiguration cost scaling factors.

cost, when the instantiation cost k_i increases, Block (0) selects larger extended intervals trying to minimize the instantiation cost derived by the dedicated capacity. Nevertheless, since for low reconfiguration cost, the system tends to allocate a higher amount of total shared capacity, AZTEC+ overall performance is similar to AZTEC as can be seen in Fig. 9a. When increasing the reconfiguration cost, AZTEC+ allocates a lower amount of total shared capacity resulting in lower normalized instantiation cost mainly due to the minimization of the dedicated capacity instantiation cost. For $k_r = 10$, there is a decreasing step between $k_i = 50$ and $k_i = 70$; allocating low amount of total shared capacity, AZTEC+ is more effective in reducing the normalized instantiation cost by dynamically selecting the extended interval T_l^* that minimizes the dedicated capacity instantiation cost.

E. Monetary Cost

We next assess the performance of AZTEC+ against three benchmarks: Traffic Predictor, a state-of-the-art mobile network traffic predictor [22]; DEEPCOG, capacity forecasting model for network resource allocation [12] and AZTEC, as the extended and improved work [23]; all solutions are based on custom-built DNNs. Traffic Predictor is a traditional demand predictor that is agnostic of all resource management costs, DEEPCOG makes anticipatory decisions on a capacity allocation that aim exclusively at minimizing the trade-off A of overprovisioning and non-serviced demands, whereas Exponential Smoothing uses an analogous setup as Traffic Predictor, but the prediction is made using the Holt-Winters algorithm [24], to showcase the performance of a non Deep Learning based framework.

Fig. 9 provides the result of the comparative evaluation among AZTEC+ and the aforementioned benchmarks, showing the overall normalized monetary cost of the anticipatory resource management against instantiation cost k_i for different reconfiguration cost k_r values. The gain of AZTEC+ compared with DEEPCOG and Traffic Predictor is clear as AZTEC+ outperforms both benchmarks in any configuration. DEEPCOG has an increment on the cost for the operator by a factor that ranges from $2.69\times$ to $4.39\times$ when the reconfiguration cost k_r is 0.1 (Fig. 9(a) and ranges from $8.91\times$ to beyond $30\times$ for higher reconfiguration costs (Fig. 9(d). As expected, the benchmark solutions suffer more when reconfiguration costs grow, which they neglect. However, even in a situation favorable to reconfiguration-agnostic approaches where such costs are small (e.g., for $k_r = 0.1$), AZTEC+ still yields a lower economic fee. Then, we can discriminate two scenarios: DEEPCOG vs. Traffic Predictor vs. Exponential Smoothing and AZTEC+ vs. AZTEC. The cluster of DEEPCOG, Traffic Predictor, and Exponential Smoothing remains constant regardless of reconfiguration. The capacity predictor of DEEPCOG avoids SLA violations by overprovisioning. At the same time, the Traffic Predictor, using MSE as a loss function, and the Exponential Smoothing tend to forecast the traffic without discriminating if the predicted value is higher (overprovisioning) or lower (SLA violation) than the truth value. Depending on the value of k_i, the instantiation cost may become higher than the SLA cost, meaning that using DEEPCOG can be more expensive than using a standard loss function or a non Deep Learning based framework for forecasting. For low reconfiguration cost ($k_r = 0.1$), AZTEC+ monetary cost is almost the same as AZTEC, being slightly worse (Fig. 9(a).

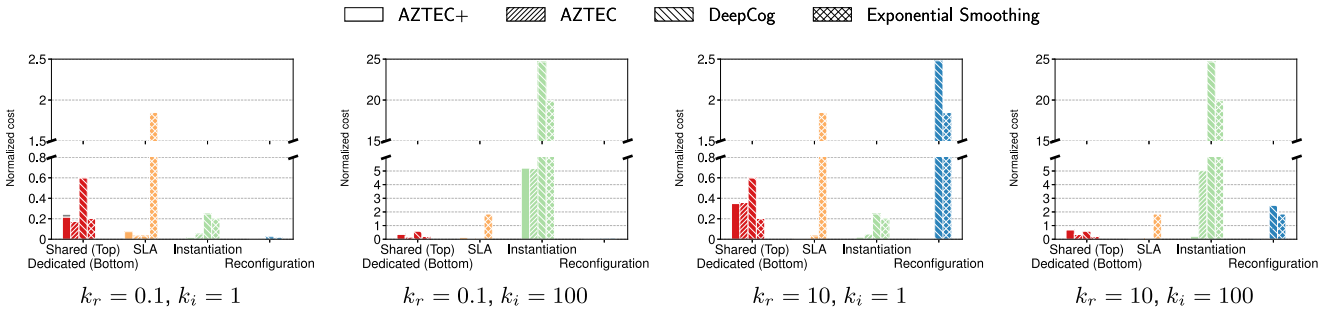

Fig. 10. Cost breakdown for AZTEC+, AZTEC, and benchmarks for the corner cases depicted in Section VI-D. The cost of shared capacity is shown in gray. The cost of dedicated capacity is shown in red at the bottom. The cost of SLA is depicted in orange, the cost of Instantiation is shown in green, and the cost of Reconfiguration is depicted in blue.

When the reconfiguration cost ($k_r = 0.5$) is increased, the normalized cost of AZTEC+ decreases substantially, using bigger extended intervals and less shared capacity. This is thanks to the dynamic selection of T_l^*, which varies depending on the instantiation cost, while for AZTEC it is set manually. From $k_r = 0.5$ (Fig. 9(b), the usage of an optimization block for selecting T_l^* in order to minimize the instantiation cost, leads to outperforming AZTEC from $k_i = 10$ onwards, in a range of $1.62\times$ to $2.27\times$. For $k_r = 1$, (Fig. 9(c) AZTEC+ outperforms AZTEC in the whole range of k_i, from $1.08\times$ to $3.38\times$. In the extreme case of $k_r = 10$ (Fig. 9(d), AZTEC+ outperforms AZTEC from $1.11\times$ to $5.85\times$. The conclusion is that depending on the reconfiguration cost, we achieve a benefit in terms of monetary cost by automatically selecting the best-extended interval. AZTEC extended interval manual selection leads to sub-optimal performance, that can be improved by leveraging the novel introduced framework, that automatically and dynamically selects the window that minimizes instantiation cost. Interestingly, the performance of a system that does not leverage a deep learning framework yields a performance similar to the deep learning based counterparts. This suggests that the pure performance of the system does not have a major impact on the overall behaviour of the system, but what makes AZTEC+ performance superior is the combination of the results coming from the three forecasting blocks, not on the specific quality of a specific predictor. For instance, DEEPCOG works substantially better than Exponential Smoothing as a pure time series predictor (the average MSE for the two cases shows a difference of almost one order of magnitude, 0.017 and 0.14 respectively). Still, the overall instantiation cost is similar for the two cases.

F. Monetary Cost Breakdown

This section analyzes how each cost impacts the extreme cases (k_r=0.1, k_i=1; k_r=10, k_i=1; k_r=0.1, k_i=100; k_r=10, k_i=100). We will compare AZTEC+ performance with the primary benchmark, AZTEC, to depict the intrinsic contribution of the different costs. Regarding the scenario with k_r=0.1 and k_i=1 (Fig. 10(a), it is important to note that we will encounter more overprovisioning costs and more SLA violation, but less instantiation cost. The overall cost is slightly higher than AZTEC. This is mainly due to a larger extended interval (21 vs 6), which means AZTEC+ experiences more overprovisioning and SLA. Despite the overall cost, having a low reconfiguration cost k_r often leads to a more extensive span of shared capacity. However, the larger window also results in more overprovisioning costs. We also achieve our primary goal, minimizing the instantiation cost. When the reconfiguration cost is high (k_r=10, k_i=1, Fig. 10(c), AZTEC+ tends to have negligible shared capacity. In this case, we have less overprovisioning cost, less SLA cost, and less instantiation cost, having a slightly bigger window size (7 vs 6). When k_r=0.1 and k_i=100 (Fig. 10(b), we observe a considerable instantiation cost. Even with a huge window, we can observe how significant the instantiation penalty is. This is due to the cost of instantiating the total shared capacity. Since k_r is low, the system allocates a large total shared capacity, which is costly to instantiate while minimizing the cost derived from the allocation of the dedicated capacity. Still, AZTEC+ provides a similar overall monetary cost. Lastly, for k_r=10, $k_i = 100$, shown in Fig. 10(d), AZTEC+ tends to have a lower amount of total shared capacity allocated due to a high reconfiguration cost. Thus, in this case, AZTEC+ minimizing the dedicated capacity instantiation cost results in minimal overall instantiation cost.

Performance of DEEPCOG and Exponential Smoothing are also included to better clarify the insights resulting from Section VI-E. DEEPCOG predictor avoids SLA violations by overprovisioning. This is confirmed in all the above scenarios, where the DEEPCOG SLA violation cost is almost 0 at the expense of overprovisioning. Instead, Exponential Smoothing does not discriminate between overprovisioning and SLA violation, leading to less dedicated resources allocated and so lower instantiation and reconfiguration costs. Even if DEEPCOG works substantially better than Exponential Smoothing as a pure time series predictor, it may result in higher or similar overall normalized cost when instantiation and reconfiguration penalties are higher.

G. Generalizability

After demonstrating that AZTEC+ cost minimization and extended interval optimization effectively outperformed the existing benchmarks, we extend our investigation to a broader urban context. The goal is to validate the scalability and adaptability of our enhanced model across different cities in Europe. Specifically, we examine the potential variances in

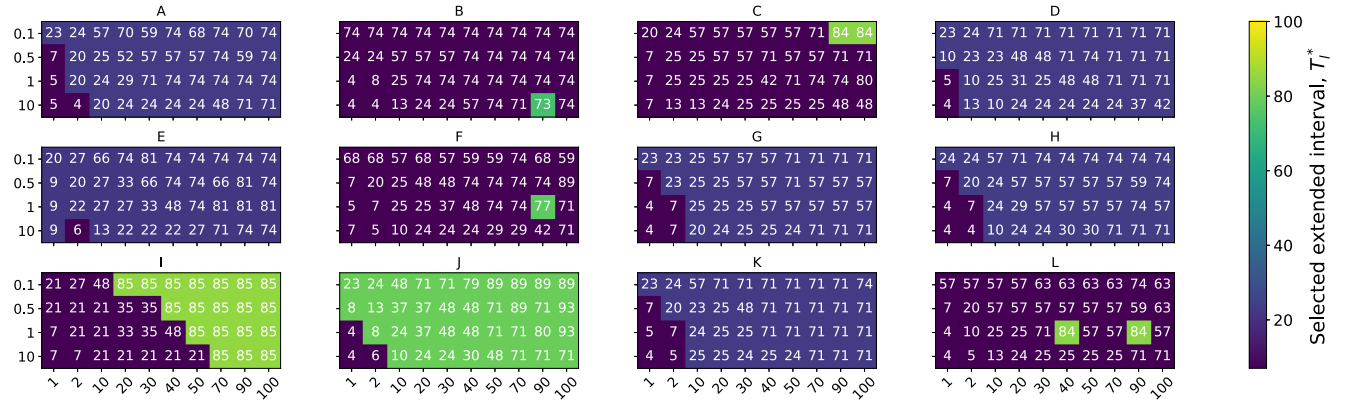

Fig. 11. Extended interval T_l^* across 12 regions encompassing urban, suburban and rural territories. For each region, we report T_l^* as a function of k_r and k_i penalties.

the Extended Interval T_l^* of our model in an heterogeneous set of cities describing urban, suburban and rural areas For each city, we deploy the AZTEC+ framework aiming to identify the values of T_l^* for the different k_i and k_r that minimize the overall cost.

Fig. 11 illustrates the selected extended interval T_l^* for various configurations of k_r and k_i for the 12 cities mentioned previously. The primary finding is that as the instantiation cost k_i increases, the model selects a longer extended interval T_l*. This is because instantiate resources becomes more expensive. As a result, the system tends to select a larger extended interval. Additionally, for any instantiation cost k_i, an increase in k_r leads to a lower selected extended interval T_l^*. In essence, we observe a trade-off between the impact of k_r and k_i. When both values are low, k_i has a greater impact on the selection of T_l^* (resulting in a larger value). If k_i remains low but k_r increases, the window T_l^* decreases. When both increase, the window continues to grow. Given the complexity of the framework and the heterogeneity of the different services' time series for each city, it is evident that some outliers deviate from the overall trend. However, it is important to note that also for these outliers selected values of T_l^* effectively minimizes the overall instantiation cost.

H. System Complexity

Fig. 12 illustrates the system's computational complexity, measured in Floating Point Operations per Section (FLOPS), as a function of the extended interval T_l^*. In our analysis, we adopt standard FLOPS formulas for both LSTM and Dense layers, consistent with those used in [25], [26].

To contextualize the theoretical complexity with practical runtime performance, all experiments were conducted on a high-performance server equipped with three NVIDIA A100 80GB PCIe GPUs, dual-socket AMD EPYC processors totaling 256 threads, and 1.4 TB of RAM. Under this configuration, the inference time for each LSTM-based block, namely, Block 1, Block Helper, and Block 2, is approximately one minute per prediction. This runtime already includes the Monte Carlo sampling procedure, in which multiple stochastic forward passes are performed using active dropout at inference time. Block 0 exhibits proportionally higher runtime due to its iterative nature, while Block 3 remains negligible in both theoretical and empirical terms.

The complexity is broken down into contributions from different system blocks, as follows. **Block (0)**: As described in Section III, its goal is to find the minimum T_l that reduces the overall cost. Then, it has to run the DNN model as many times as needed to find T_l^*. Specifically, we observed that it needs 10 iterations to find T_l^*. **Block (I)** executes forecasts every T_l^* steps. Its complexity decreases as T_l^* increases due to less frequent execution. **Block (II)**, similar to Block (I), it also forecasts at interval T_l^* and also contributes less to the aggregated FLOPS due to its architecture, described later. **Block (III)** is represented but not visualized due to its negligible contribution in comparison with DNN architecture as it is a classical optimization using BOBYQA (Section V). **Helper** performs forecasting at every individual timestep, leading to a high and constant computational load that does not depend on T_l^*.

As described in Fig. 4, each block uses LSTM-based neural networks excluding Block (III). Block (0), (1) and the helper share the same architecture, comprising an LSTM layer of 128 cell units followed by dropout and Dense layers of 64 and 5 (*e.g.*, number of slices). Block (II) uses a simplified architecture with the same LSTM layer and a Dense layer of 1. Thus, even though the LSTM layers have more impact in terms of complexity compared to Dense layers, we observe a difference between both architectures. Importantly, the system incorporates Monte Carlo sampling to estimate uncertainty and the robustness of predictions. The use of multiple stochastic forward passes increases the computational cost, especially for blocks executed more frequently. Overall, we observe that as the extended interval T_l^* increases, the total computational cost decreases due to the less frequent execution of Block (I) and (II). Yet, Block (0) and the helper stay the blocks with highest complexity. Block (III) complexity is almost negligible.

VII. Related Work

The use of AI and ML is taking a leading role in the management, orchestration, and operation of networks. The interested reader may find comprehensive surveys

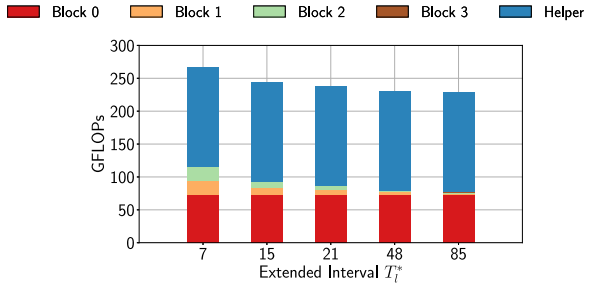

Fig. 12. Computational complexity of the AZTEC+ framework for different extended interval durations T_l^*. The complexity is expressed in GFLOPs.

in [3], [27], [28]. Specifically, AZTEC+ aims to provide an automated solution for resource orchestration in virtualized networks. The capability of predicting the future traffic demand through ML algorithms is on the basis of several recent works in the literature. For instance, [29] uses Holt-Winters Forecasting to perform short-term prediction of the network slices load and optimize radio scheduling. Another work that tackles the problem of mobile traffic forecasting is the one in [30], which employs a decomposition approach to perform an ARIMA forecasting of the time series. A similar solution was also described in [31]. However, deep learning solutions are becoming more and more popular in the state-of-the-art solutions for autonomous network orchestration as, due to their versatility, they can be applied to a wide scope of problems, ranging from radio-related [32] to service orchestration ones [33]. Besides the solutions already analyzed in Section VI, other recent efforts for deep-learning-based mobile traffic forecasting propose advanced DNN architectures to achieve long-term forecasting with a relatively small input data [34].

Still, none of the aforementioned solutions consider monetary implications when forecasting mobile traffic demand. This concept was originally introduced by [12], which, however, does not take into account important aspects such as instantiation and reconfiguration costs, as already discussed in Section III. Finally, [35] proposed a framework combining Deep Learning and Optimization for maximizing the profit using a cost-driven orchestration doing overbooking of network slices. The problem is formulated with a modular SLA function, and depending on its shape, the problem can be Mixed-Integer Nonlinear Programming (MINLP) or Mixed-Integer Linear Programming (MILP). Hence, it clearly shows the advantages of automatic resource re-orchestration. This further motivates the need for a zero-touch system that optimizes costs through efficient resource utilization while considering the orchestration framework characteristics (*i.e.*, VM boot-up times, network re-configurations), as AZTEC+ does. Thus, the framework proposed in AZTEC+ has unique characteristics compared to the landscape of works available in the literature. Our solution, which involves predictors working at different timescales.

VIII. Conclusion

In the context of ever-increasing complex networks and the need of zero-touch management AZTEC+ demonstrates the viability of automatically optimizing the different cost trade-offs that happen in a virtualized mobile network, to minimize resource provisioning. In detail, AZTEC+ automatically selects resource orchestration intervals based on past service-level demands. We evaluate AZTEC+ with data coming from a large-scale production network over different cities in France. Our experiment shows that, depending on the different cost factors contributing to the overall network, AZTEC+ outperforms the different benchmarks by a factor up to $5.85\times$.

References

[1] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "How should I slice my network?: A multi-service empirical evaluation of resource sharing efficiency," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (ACM MobiCom)*, 2018, pp. 191–206.

[2] V. Sciancalepore et al., "A future-proof architecture for management and orchestration of multi-domain NextGen networks," *IEEE Access*, vol. 7, pp. 79216–79232, 2019.

[3] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287,3rd Quart., 2019.

[4] B. Koley, "The zero touch network," in *Proc. 12th Int. Conf. Netw. Service Manage. (IEEE CNSM)*, 2016.

[5] *ZSM Scenarios and Key Requirements*, ETSI Standard ISG ZSM 001, Oct. 2018.

[6] A. Garcia-Saavedra, X. Costa-Pérez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vRAN/MEC orchestration," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 2366–2374.

[7] J. Kim, D. Kim, and S. Choi, "3GPP SA2 architecture and functions for 5G mobile communication system," *ICT Exp.*, vol. 3, no. 1, pp. 1–8, Mar. 2017.

[8] L. Peterson et al., "Central office re-architected as a datacenter," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[9] J. Hao, T. Jiang, W. Wang, and I. K. Kim, "An empirical analysis of VM startup times in public IaaS clouds," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, 2021, pp. 398–403.

[10] "Refined design of 5G-CORAL orchestration and control system and future directions." 5G-CORAL, Public Deliverable. May 2019. [Online]. Available: https://euprojects.netcom.it.uc3m.es/5g-coral/deliverables/

[11] S. González, A. De la Oliva, C. J. Bernardos, and L. M. Contreras, "Towards a resilient Openflow channel through MPTCP," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BSMB)*, 2018, pp. 1–5.

[12] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 280–288.

[13] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Pérez, "Overbooking network slices through yield-driven end-to-end orchestration," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol. (ACM CoNEXT)*, 2018, pp. 353–365.

[14] L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, and X. Costa-Pérez, "OVNES: Demonstrating 5G network slicing overbooking on real deployments," in *Proc. IEEE Int. Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2018, pp. 1–2.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. Cambridge, MA, USA: Cambridge Univ. Press, 2007.

[16] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Alpha-OMC: Cost-aware deep learning for mobile network resource orchestration," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM)*, 2019, pp. 423–428.

[17] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2013, pp. 8609–8613.

[18] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *Schedae Informaticae*, vol. 25, pp. 49–59, Mar. 2017.

[19] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1050–1059.

[20] M. J. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives," Dept. Appl. Math., Univ. Cambridge, Cambridge, U.K., Rep. NA2009/06, 2009.
[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*. New York, NY, USA: Cambridge Univ. Press, 1993, vol. 2.
[22] J. Wang et al., "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.
[23] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "AZTEC: Anticipatory capacity allocation for zero-touch network slicing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2020, pp. 794–803.
[24] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. Melbourne, VIC, Australia: OTexts, 2018.
[25] Y. Yang et al., "Accelerating and compressing LSTM based model for online handwritten Chinese character recognition," in *Proc. 16th Int. Conf. Front. Handwrit. Recognit. (ICFHR)*, 2018, pp. 110–115.
[26] M. Zhang, W. Wang, X. Liu, J. Gao, and Y. He, "Navigating with graph representations for fast and scalable decoding of neural language models," in *Proc. 32nd Adv. Neural Inf. Process. Syst.*, 2018, pp. 1–12.
[27] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.
[28] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.
[29] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.
[30] F. Xu et al., "Big data driven mobile traffic understanding and forecasting: A time series approach," *IEEE Trans. Services Comput.*, vol. 9, no. 5, pp. 796–805, Sep./Oct. 2016.
[31] Y. Akinaga, S. Kaneda, N. Shinagawa, and A. Miura, "A proposal for a mobile communication traffic forecasting method using time-series analysis for multi-variate data," in *Proc. IEEE Glob. Telecommun. Conf. (GLOBECOM)*, 2005, p. 6.
[32] C. Luo, J. Ji, Q. Wang, X. Chen, and P. Li, "Channel state information prediction for 5G wireless communications: A deep learning approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 227–236, Jan.–Mar. 2020.
[33] B. Li, W. Lu, S. Liu, and Z. Zhu, "Deep-learning-assisted network orchestration for on-demand and cost-effective VNF service chaining in inter-DC elastic optical networks," *J. Opt. Commun. Netw.*, vol. 10, no. 10, pp. D29–D41, Oct. 2018.
[34] C. Zhang, M. Fiore, and P. Patras, "Multi-service mobile traffic forecasting via convolutional long short-term memories," in *Proc. IEEE Int. Symp. Meas. Netw. (IEEE M N)*, 2019, pp. 1–6.
[35] S. Alcalá-Marín, A. Bazco-Nogueras, A. Banchs, and M. Fiore, "kaNSaaS: Combining deep learning and optimization for practical overbooking of network slices," in *Proc. 24th Int. Symp. Theory, Algorithm. Found., Protocol Design Mobile Netw. Mobile Comput.*, 2023, pp. 51–60.

Sergi Alcalá-Marín received the M.Sc. degree in advanced telecommunications technologies from the Universitat Politècnica de Catalunya, Spain, in 2020, and the Ph.D. degree in telematics engineering from the University Carlos III of Madrid and IMDEA Networks Institute in 2025. He is a Research Scientist with the University Carlos III of Madrid. His research interests include wireless communications, network automation, and AI-driven network optimization.

Dario Bega (Member, IEEE) received the M.Sc. degree in telecommunications engineering from the University of Pisa, Italy, in 2013, and the Ph.D. degree in telematics engineering from the University Carlos III of Madrid and IMDEA Networks Institute in 2020. He is a Network System Automation Researcher with Nokia Bell Labs. His research interests include network automation and AI-driven network optimization.

Marco Gramaglia received the Ph.D. degree in telematics engineering from the Universidad Carlos III de Madrid. He has contributed extensively to several research projects at both the European and national levels. He co-authored more than 100 articles and, according to Google Scholar, his H-index is 35. His research interests include network automation, privacy, and AI-driven resource management.

Albert Banchs (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees from the Polytechnic University of Catalonia (UPC-BarcelonaTech) in 1997 and 2002, respectively. He is currently a Full Professor with the University Carlos III of Madrid (UC3M), with double affiliation as the Director of the IMDEA Networks Institute. Before joining UC3M, he was with ICSI Berkeley in 1997, Telefonica I+D in 1998, and NEC Europe Ltd., from 1998 to 2003. He was an Academic Guest with ETHZ in 2012, a Visiting Professor with EPFL in 2015, 2013, and 2018, and a Fulbright Scholar with The University of Texas at Austin in 2019. He is the author over 150 publications in international conferences and journals and is the co-inventor of several patents.

Xavier Costa-Perez (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in telecommunications from the Polytechnic University of Catalonia, Barcelona. He is an ICREA Research Professor, a Scientific Director of the i2cat Research Center, and the Head of 6G R&D with NEC Laboratories Europe. His team generates research results that are regularly published at top scientific venues, produces innovations that have received several awards for successful technology transfers, and participates in major European Commission research and development collaborative projects. He has held multiple leadership positions both in industry and research organizations, such as deputy general manager, chief researcher, technology board member, and scientific advisory board member. As a standards delegate, he contributed to multiple standardization bodies (e.g., IEEE 802.11, 802.16, WiFi Alliance, and 3GPP) and was recognized in several standards as a top contributor. He was the recipient of a national award for his Ph.D. thesis. He has served on the organizing committees of several conferences (including ACM MOBICOM, IEEE INFOCOM, WCNC, and Greencom), published papers of high impact, and holds about 100 granted patents. He has served as an Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON COMMUNICATIONS, and *Computer Communications* (Elsevier).

Marco Fiore (Senior Member, IEEE) received the M.Sc. degrees from the University of Illinois at Chicago and the Politecnico di Torino, the Ph.D. degree from the Politecnico di Torino, and the Habilitation à Diriger des Recherches from the Université de Lyon. He is a Research Professor with IMDEA Networks Institute, where he leads the Networks Data Science Group, and the Co-Founder and the CTO of Net AI. He has held tenured positions with the Institut National des Sciences Appliquées de Lyon and the National Research Council of Italy, and has been a Visiting Researcher with Rice University, the Universitat Politècnica de Catalunya, and University College London. His research is at the interface of mobile networks and data science, and has received funding from the European Commission and national agencies in Spain, France, and Italy, as well as a number of recognitions that include two best paper awards at IEEE INFOCOM. He is a former Marie Curie Fellow and a Royal Society Visiting Research Fellow, and a Senior Member of ACM.