I. ML-Driven SLA Management for OPEX Minimization

In the previous section, slice sharing was discussed as a use case of ML-driven provisioning and management of vertical slices, using the AIML platform of 5Growth at the VS level. In this section, we present SLA management for OPEX minimization as the second use case of the platform which is conducted at the SO level.

After describing the internal building blocks in the 5Gr-SO that are related with AIML-based SLA assurance, the the problem and the proposed solution are discussed.

The ultimate goal of the Communication Service Providers (CSPs) is to maximize the profit via minimizing the OPEX. Three factors mainly contribute to the cost:

- The cost of allocating resources for the services, i.e., the cost of instantiating VNFs such as license cost.
- The cost of operating the services, i.e., the cost of keeping the VNF instances up and running such as energy cost.
- The SLA violation cost, i.e., the penalty that the CSP should pay if the QoS level, which is agreed in the SLA between the CSP and the customer, is violated.

There are two challenges to achieve the goal. First, these costs are conflicting; to minimize the SLA violation cost, CSP should allocate sufficiently large resources to handle the peak load of the service that will significantly increase the instantiation and operation cost. And vise versa, if the CSP aims to minimize the provisioning and operation cost by allocating the minimum resources to the services, it will incur a considerable SLA violation cost. So, the problem is to find the optimal trade-off between the costs, i.e., the *optimum IL* that minimizes the OPEX. The second challenge is that the optimum IL depends on the traffic load which is time-varying; so the CSP should *scale* up/down the VNFs over time.

The ML-driven approach for provisioning and management services in the 5Growth platform can be utilized to tackle the complexities efficiently in a similar way discussed in the slice sharing use case. The overall procedure is as follows. 5Gr AIMLP trains and provides the model that, according to the current IL, the current level of QoS, etc., determines the appropriate IL to satisfies the SLA. 5Gr-SO periodically collects the monitoring data and utilizes the model to find the appropriate IL; then the SLA manager computes the target IL to make the trade-off between the costs. In the following sections, first, we discuss the assumptions and system model; then, we elaborate the ML-driven SLA management solution.

A. Assumptions and System Model

In this problem, service latency is the QoS metric that is agreed in the SLA. We assume multi-level SLA management where a set $\Delta = \{\delta_1, \dots, \delta_L\}$ of delay thresholds where $\delta_i < \delta_{i+1}$, and their associated set of penalties $\mathcal{P} = \{p_1, \dots, p_L\}$ where $p_i \ll p_{i+1}$ are specified in the SLA. Let τ be the service latency, if the SLA is violated, the cost is

$$\sigma_{\mathsf{sla}} = p_{\bar{l}}(\tau - \delta_{\bar{l}}), \tag{1}$$

where $\bar{l}$ is the largest thresholds that is violated, i.e.,

$$\bar{l} = \underset{l}{\operatorname{argmin}} \max\{0, (t_s - \delta_l)\}.$$

This is a suitable model for the applications that support multiple levels of QoS such as video streaming.

The network services are composed of a number of VNFs; but it is assumed that there is a bottleneck VNF that dominates the service latency, as it is justified by the applications discussed in Section **??**. So, to comply with the SLA, the CSP only needs to scale up/down the bottleneck VNF. It is assumed that there is a set $\mathcal{D}$ of the NSDs (Virtual Network Function Descriptor) of the bottleneck VNF where each descriptor $d \in \mathcal{D}$, besides the other settings, specifies n_d as the number of the instances of the VNF. When the SLA manager decides to switch from descriptor d to d' where $n_d < n_{d'}$, the SO creates $n_{d'} - n_d$ new instances where the cost of creating a new instance of the VNF is σ_{ins}; and the cost of using an instance per unit of time is σ_{opr}. There is no cost for termination.

Consider a time period T, define $\mathcal{R}$ as the set of requests arrive in this period, and $\mathcal{C}$ as the set of instances created during this period; moreover, let t_c^i and t_t^i be respectively the instantiation and termination times of the instance i. The objective is to minimize the OPEX defined as

$$OPEX = \sum_{r \in \mathcal{R}} \sigma^r_{\mathsf{sla}} + \sum_{i \in \mathcal{C}} \big(\sigma_{\mathsf{ins}} + \sigma_{\mathsf{opr}}(t_t^i - t_c^i)\big). \tag{2}$$

To achieve it, as mentioned, the CSP needs to find the optimum number of instances, i.e., the best NSD at each time $t \in \mathcal{T}$ that decreases the first term of the objective function but does not increase the second term significantly. The problem is not trivial because the optimum IL depends on not only the current traffic load but also the future dynamic of the load which is not known. For example, the no scaling strategy, i.e., using a fixed descriptor independent of the traffic load, can lead to a significant cost as if the load is greater than the capacity of the instances, there will be a large SLA violation cost and if the load is less than the capacity, the second term of the objective function will be considerable. As another example, fast switching between the descriptors according to just the current status of the network and SLA also increases OPEX as it causes too many instantiation and termination of VNFs that significantly increases $\sum_{i \in \mathcal{C}} \sigma_{\mathsf{ins}}$ in the objective function. In the following section, we develop an ML-driven solution that is based on the 5Gr-SO workflows.

B. VNF Resource Scaling Solution

The solution presented in this section is based on the 5Gr AIMLP and its interaction with the other entities of the platform as explained in Section **??** and also the workflow of 5Gr-SO detailed in Section **??**. So, all the details are not repeated here. The overall solution is also similar to the slice sharing use case, i.e., the VNF resource scaling algorithm, which is a logic in the SLA Manager of 5Gr-SO, uses an ML model, already trained and maintained by 5Gr AIMLP, to determine the suitable NSD of the VNF. However, there are three important differences. First, this solution does not run

upon the arrival of each request; indeed, it runs periodically. Second, the ML model used in this solution does not determine the latency class of requests, the output of the model is the NSD to satisfy the QoS requirement of the service. Third, the algorithm aims to minimize the OPEX rather than maximizing the instance sharing.

In the training phase, 5Gr AIMLP trains the ML model using the dataset where the features are i) CPU utilization u_{cpu}, ii) memory utilization u_{ram}, iii) service latency τ, and iv) current NSD d; the label is the target NSD to satisfy the QoS requirement of the service. Then in the operation phase, periodically, the following steps are performed in j-th period:

1) The monitoring data u^j_{cpu}, u^j_{ram}, τ^j, and d^j are collected.
2) The exponential moving averages of the monitoring data are updated as $\bar{x} \leftarrow \alpha x^j + (1-\alpha)\bar{x}$.
3) The ML model runs using the averages $\bar{u}^j_{\text{cpu}}$, $\bar{u}^j_{\text{ram}}$, $\bar{\tau}$, and d^j that provides d^{j+1}_{ml}.
4) The VNF Resource Scaling (VRS) algorithm runs using d^{j+1}_{ml} and determines d^{j+1}.
5) If $d^j \neq d^{j+1}$, the SLA Manager triggers the scaling operation.

As discussed, fast switching between NSDs while can decrease σ_{sla} and σ_{opr}, it incurs significant instantiation cost $\sum_{i\in\mathcal{C}} \sigma_{\text{ins}}$. To alleviate it, three steps are taken into account in this solution. First, the monitoring period is much larger than the request inter-arrival time to avoid triggering the scaling operation per request. Second, instead of the instantaneous monitoring data, the exponential moving average is used as the input of the model. Third, the VRS algorithm, presented in Algorithm 1, takes into account the SLA violation cost, instantiation cost, and the usage cost to obtain the target NSD.

This algorithm, in addition to the descriptor suggest by the model, d^{j+1}_{ml}, takes the SLA violation and operation costs in this monitoring interval, which are respectively denoted by Σ^j_{sla} and Σ^j_{opr}. It also takes the scaling direction SD^{j-1} suggest by the model in the previous interval; $SD = 0$ implies no scaling, $SD > 0$ means scaling up and vice versa. The algorithm at the beginning (Line 1) finds the scaling direction suggest by the ML model in this interval. If it is "scale up" but the model has changed the direction, in line 4, the SLA violation cost of this interval is saved as the accumulated SLA violation cost Σ_{sla}. However, if the model is continuously requesting scale up, the accumulated SLA violation cost is updated and if it is large enough that implies it is beneficial to create new instances to decrease the violation cost, then, the suggest descriptor is selected as the target descriptor (Line 10). In a similar way, when the model suggests scaling down (Line 11), this algorithm checks it and decides to scale down only if the previous suggestion was also scaling down and the operation cost is large enough.

The conditions in lines 7 ad 16 aims to make the trade-off between the costs where $0 < \beta < 1$ aNSD $0 < \gamma < 1$ are tunable parameters depends on the dynamics of the traffic load. If the traffic load is highly dynamic, these parameters should be large to avoid too many instantiations and terminations by small changes in the traffic load; however, if traffic load changes smoothly, the value of the parameters can be small to minimize the SLA violation cost and the operation cost.

Algorithm 1 VNF Resource Scaling (VRS)

Require: $d^{j+1}_{\text{ml}}, \Sigma^j_{\text{sla}}, \Sigma^j_{\text{opr}}, SD^{j-1}$
1: $SD^j \leftarrow \text{sign}(n_{d^j} - n_{d^{j+1}_{\text{ml}}})$
2: **if** $SD^j > 0$ **then** ▷ scaling up suggestion by ML model
3: **if** $SD^{j-1} \not> 0$ **then** ▷ a new scaling up suggestion
4: $\Sigma_{\text{sla}} \leftarrow \Sigma^j_{\text{sla}}$
5: **else** ▷ ML model keeps requesting scaling up
6: $\Sigma_{\text{sla}} \leftarrow \Sigma_{\text{sla}} + \Sigma^j_{\text{sla}}$
7: **if** $\Sigma_{\text{sla}} > \beta\sigma_{\text{ins}}$ **then** ▷ large SLA violation cost
8: $d^{j+1} \leftarrow d^{j+1}_{\text{ml}}$
9: **else**
10: $d^{j+1} \leftarrow d^j$
11: **else if** $SD^j < 0$ **then** ▷ scaling down suggestion by ML
12: **if** $SD^{j-1} \not< 0$ **then** ▷ a new scaling down suggestion
13: $\Sigma_{\text{opr}} \leftarrow \Sigma^j_{\text{opr}}$
14: **else** ▷ ML model keeps requesting scaling down
15: $\Sigma_{\text{opr}} \leftarrow \Sigma_{\text{opr}} + \Sigma^j_{\text{opr}}$
16: **if** $\Sigma_{\text{opr}} > \gamma\sigma_{\text{ins}}$ **then** ▷ large operation cost
17: $d^{j+1} \leftarrow d^{j+1}_{\text{ml}}$
18: **else**
19: $d^{j+1} \leftarrow d^j$
20: **return** d^{j+1}

C. Simulation Results

In this section, we evaluate the performance of the SLA management solution proposed for OPEX minimization. For this purpose, we consider the automotive services explained in **??**. It is assumed that the arrival and departure of requests are Poisson processes where the average arrival rate is time-varying over 24 hours period. It increases from 4:00 to 10:00, decreases a bit from 10:00 to 15:00, then increases again until 22:00, and after that decrease again. The training dataset is labeled by three labels: "small NSD", "big NSD" and "no change"; so, $\mathcal{D} = \{\text{Small-NSD}, \text{Big-NSD}\}$ and it is assumed that $d_{\text{Small-NSD}} = 2$, and $d_{\text{Big-NSD}} = 5$. We consider two-level SLA, i.e., $\Delta = \{\delta_1, \delta_2\}$ and $\mathcal{P} = \{p_1, p_2\}$. In these simulations, three strategies are compared: the L-INS and H-INS strategies that respectively instantiate 2 and 5 instances at the beginning and don't change it later, moreover, the ML-RS strategy which is the proposed solution.

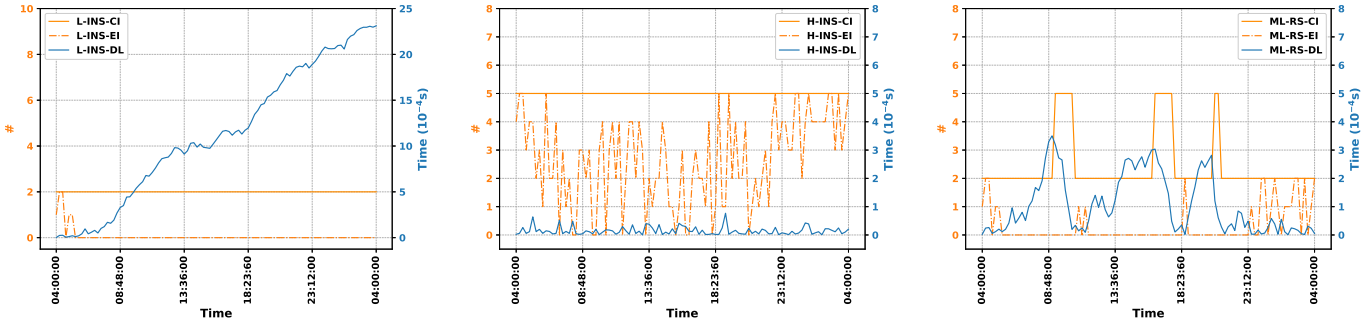
Fig. 1 shows traces of the operation of the strategies for a period of time. In this figure, the number of created instances, the number of empty instances, and the service latency of each strategy are respectively postfixed by "-CI", "-EI", and "-DL". It is seen that, while the small number of instances by L-INS are always busy but they are not sufficient to serve all the requests, and consequently, the processing latency grows exponentially. Contrarily, the large number of instances by H-INS effectively decreases the processing latency but there is a

considerable number of empty instances. On the other hand, the ML-RS strategy efficiently keeps the processing latency under the SLA thresholds, which $\delta_1 = 14.7$ ms and $\delta = 26.2$ ms in these simulations, and does not wast empty instances. This efficient scaling results in significant OPEX minimization which is depicted in Fig. 2.

In Fig 2, the SLA violation cost, the service provisioning cost, which is the sum of instantiation and operation costs, and also the OPEX are shown per scaling strategy. These results show that while L-INS has the minimum provisioning cost, due to the significant SLA violation, the OPEX is the largest one. In the H-INS, the SLA violation is almost zero but because of the high operation cost, again the OPEX is significant. The ML-RS has the minimum OPEX via appropriate resource scaling. The jumps in the OPEX are because of switching from the small NSD to the big NSD.

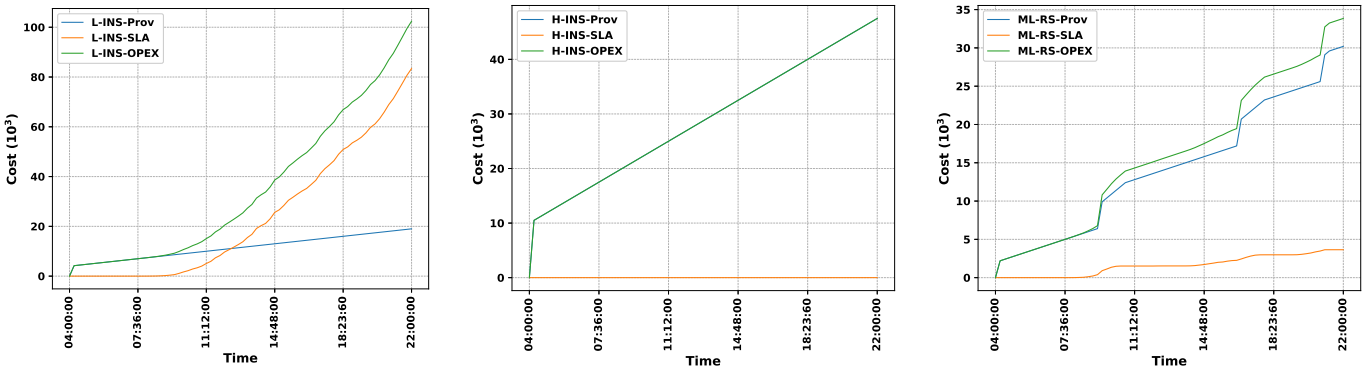
To evaluate the performance of the strategies, they are compared with respect to the traffic load in Fig. 4. In these results, in each experiment, the arrival rate of the requests is multiplied by parameter ℓ and the total cost is computed over 24 hours period in 4-(a),(b) and over 48 hour period in 4-(c). These are the average results of 20 experiments per ℓ. As it is shown, in the lightly loaded conditions, $\ell \leq 0.3$, all strategies comply with the SLA, so there is no SLA violation cost and because of over-provisioning, the OPEX of H-INS is significant while L-INS and ML-RS use the minimum number of instances and have the same OPEX. By increasing ℓ, the SLA violation cost of L-INS grows exponentially but the proposed solution can maintain a negligible SLA violation by efficient resource scaling that minimizes the OPEX. In the highly loaded conditions $\ell > 1$, even the instances in the big descriptor are not sufficient to handle the offered load, so the proposed solution and the H-INS has a similar performance.

To summary, these results show the superiority of the ML-driven VNF resource scaling solution to simultaneously manage the SLA of requests and minimize the OPEX. The proposed solution demonstrates how the ML-driven algorithms can be deployed via the 5Growth platform.



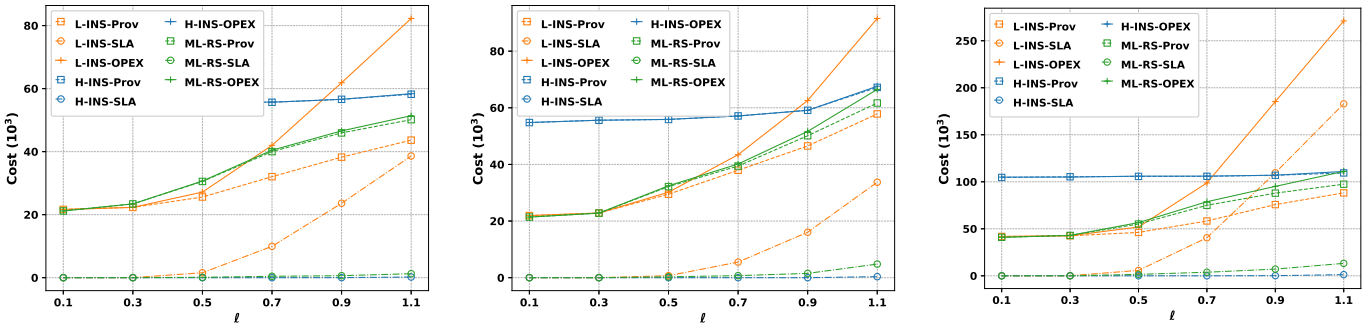
(a) L-INS Strategy (b) H-INS Strategy (c) ML-RS Strategy

Fig. 1: The traces of operation of the scaling strategies that show the number of created instances, the number of empty instances, and the processing latency.



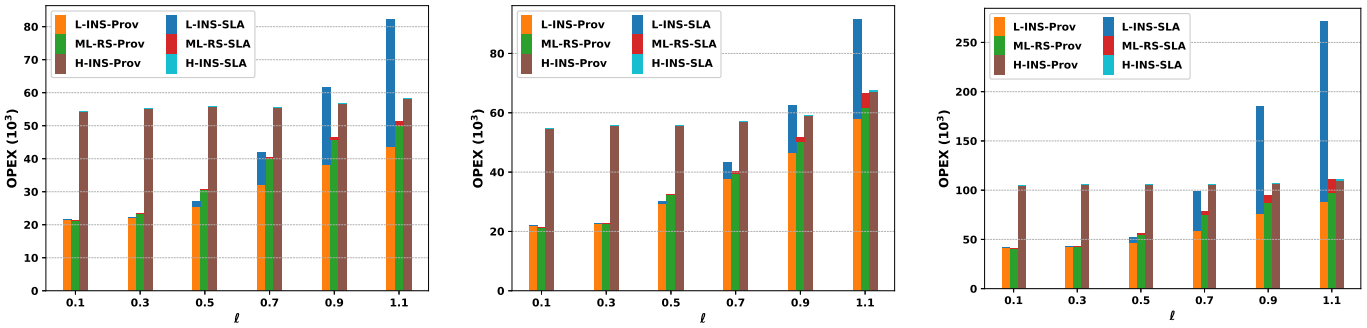
(a) L-INS Strategy (b) H-INS Strategy (c) ML-RS Strategy

Fig. 2: The traces of cost of the scaling strategies that show the service provisioning cost, SLA violation cost, and the OPEX.



(a) Collision Detection services in a 24 hour period (b) Bird Eye service in a 24 hour period (c) Bird Eye service in a 48 hour period

Fig. 3: The Provisioning cost, SLA violation cost and the OPEX with respect to the scale of the arrival rate of the requests.



(a) Collision Detection services in a 24 hour period (b) Bird Eye service in a 24 hour period (c) Bird Eye service in a 48 hour period

Fig. 4: The Provisioning cost, SLA violation cost and the OPEX with respect to the scale of the arrival rate of the requests.