SLA Decomposition for Network Slicing: A Deep Neural Network Approach

Cyril Shih-Huan Hsu, Danny De Vleeschauwer* and Chrysa Papagianni

***Abstract*—For a network slice that spans multiple technology and/or administrative domains, these domains must ensure that the slice's End-to-End (E2E) Service Level Agreement (SLA) is met. Thus, the E2E SLA should be decomposed to partial SLAs, assigned to each of these domains. Assuming a two-level management architecture consisting of an E2E service orchestrator and local domain controllers, we consider that the former is only aware of historical data of the local controllers' responses to previous slice requests, and captures this knowledge in a risk model per domain. In this study, we propose the use of Neural Network (NN) based risk models, using such historical data, to decompose the E2E SLA. Specifically, we introduce models that incorporate monotonicity, applicable even in cases involving small datasets. An empirical study on a synthetic multi-domain dataset demonstrates the efficiency of our approach.**

***Index Terms*—network slicing, service level agreement, risk model, quality of service, deep neural network**

I. Introduction

Continuing the trend from 5G systems, the future mobile network is expected to be a multi-service network supporting multiple vertical industries with a diverse set of requirements. Network slicing, introduced in 5G, enables the creation and operation of multiple logical networks over the shared infrastructure, tailored to the requirements of services with agreed upon SLAs. SLAs are provider-customer contracts setting the expected quality, performance, and availability of a service in terms of measurable targets such as throughput, latency, reliability etc., known as Service-Level Objectives (SLOs).

A network slice may span different parts of the network (i.e., access, transport and core network) and could be deployed across multiple infrastructure providers. The deployment of the E2E service across a set of domains must meet the agreed upon SLOs. To this end, the E2E SLA associated with the slice should be decomposed into partial SLOs assigned and supported by each of these domains. Decomposing the E2E SLA in the underlying domains' requirements is an inevitable step in resource allocation [1]. AI-assisted SLA decomposition is considered key to automating 6G complex business processes [2].

Similar to [3], we assume a two-level management architecture consisting of an E2E service orchestrator responsible for network service lifecycle management, and local domain controllers that are in charge of instantiating parts of the network slice in their respective domains (Fig. 1). The orchestrator determines the SLA decomposition for the incoming service request, while the domain controllers perform admission control within their domains, prior to resource allocation required for E2E slice deployment. We assume that the orchestrator has no knowledge of the state of the infrastructure at the moment that SLA decomposition is performed. However, we consider that admission control information (i.e., request acceptance or rejection) from each domain is accessible to the orchestrator. Thus, it can make informed decisions using domain-specific risk models employing such data.

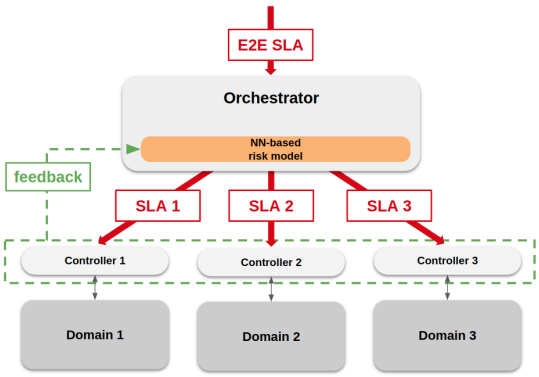


Fig. 1: Network slicing management and orchestration system

Existing approaches for SLA decomposition mainly employ heuristics, considering state information periodically reported by the underlying domains [1]. Authors in [4] propose an E2E SLA decomposition system, using supervised machine learning algorithms. In our previous work [3], the problem is addressed in three steps: (i) we formulate the decomposition problem as a function of the SLA acceptance probability per domain, under the constraints set by the E2E SLOs; (ii) we use a parameter-free risk model per domain to estimate these probabilities, constructed by observing the response of domain controllers to previous requests. We formulate the risk model estimation as a likelihood maximization problem, under monotonicity constraints stemming from the nature of the decomposition problem and employ Sequential Quadratic Programming (SQP) to solve it. (iii) Using the risk models, we apply an initial exhaustive search followed by SQP to solve the decomposition problem for each request. However, the estimation of a parameter-free risk model is non-linear, hence computationally intensive.

In this paper we introduce risk models based on deep NNs to support the SLA decomposition process. In general, we want to make as few assumptions as possible to obtain

Cyril Shih-Huan Hsu and Chrysa Papagianni are with Informatics Institute, University of Amsterdam, The Netherlands (email: s.h.hsu@uva.nl, c.papagianni@uva.nl).

*Danny De Vleeschauwer is with Nokia Bell Labs, Antwerp, Belgium (e-mail: danny.de_vleeschauwer@nokia-bell-labs.com.

Work partially funded by EC-funded projects H2020 DAEMON (grant no. 101017109) and HORIZON SNS JU DESIRE6G (grant no. 101095890).

the risk model per domain [3]. To this end, we propose methods that allow NN-based risk models to approximate monotonic functions without compromising their expressiveness. Most of these models perform robustly even with small datasets. Furthermore, they are capable of learning from a more complex dataset with a constant inference time, thus SLA decomposition improves accordingly, increasing the accuracy and scalability of the approach. Regarding the time required for model construction, it scales linearly with the number of samples, while the approach in [3] scales polynomially. This make them also appropriate for online-learning risk models, enabling adaptation in a more dynamic environment.

Section II provides an overview of the SLA decomposition problem. In Section III we present the proposed deep NN-based risk models. In Section IV we describe the experimental setup while in Section V we evaluate the problem using the proposed models. Section VI presents our conclusions.

II. Problem Description

A. SLA Decomposition

The E2E SLA s_{e2e} is described as a vector of SLOs with respect to certain performance metrics [3]. The values of the SLO vector represent the constraints defined in SLA in order; for example, an SLA comprises an E2E delay and throughput, i.e., $s_{e2e} = (\tau_{e2e}, \theta_{e2e})$. This vector indicates that the corresponding network slice has to be operated in the way that its E2E delay τ and throughput θ fulfill the constraints given by the SLO vector, i.e., $\tau \leq \tau_{e2e}$, and $\theta \geq \theta_{e2e}$. Assuming that the network slice spans N domains, where $n = \{1, 2, ..., N\}$, we define the vector s_n that represents the n-th domain SLOs, and the equation $s_{e2e} = G(s_1, s_2, ..., s_N)$ that describes how per domain SLOs form the E2E objective s_{e2e}. For instance, the E2E delay is the sum of all delays for the involved domains, and the E2E throughput is the minimum throughput from all domains, i.e., $\tau_{e2e} = \Sigma_{n=1}^{N} \tau_n$ and $\theta_{e2e} = \min\{\theta_1, \theta_2, ..., \theta_N\}$.

The ability of a domain to support a partial SLA s_n, is captured by a risk model. The risk models of all involved domains can be used in the SLA decomposition process. The risk model is defined as $-\log P_n(s_n)$, where $P_n(s_n)$ represents the probability that a request in the n-th domain with s_n SLOs is accepted. Assuming each domain makes decisions independently, then the E2E acceptance probability is the product of the acceptance probabilities of all individual domains. Therefore, the E2E decomposition can be formulated as an optimization problem that minimizes the objective (1) under the constraints (2) as follows:

$$\min_{\mathbf{s}} - \sum_{n}^{N} \log P_n(s_n) \tag{1}$$

$$\text{s.t. } s_{e2e} = G(s_1, s_2, ..., s_N) \tag{2}$$

Furthermore, if a parameterized neural network $\mathcal{F}_n$ is utilized as the risk model, where $\mathcal{F}_n(s_n) = P_n(s_n)$, the objective function in Eq. 1 can be rewritten as:

$$- \sum_{n}^{N} \log \mathcal{F}_n(s_n) \tag{3}$$

The goal of the decomposition policy is to minimize the overall risk of rejecting the corresponding request with SLOs s_n for each participant n-th domain.

B. Determining Neural Network-based Risk Models

Based on our previous work [3], we determine the risk model per domain. Given a new request with specific SLOs s (we omit the domain subscript n as we focus on a single domain), a controller has to decide whether or not to accept the request. To make such decision, the controller has to consider also the state ω of the infrastructure at the moment. This state is defined by the utilization and loads on the links and servers, the delays incurred over network links and the calculated backup paths, etc. It is also determined by the randomness of the SLAs of all previous incoming requests as well as the decisions taken by the domain. Therefore, certain (s, ω) pairs will lead to acceptance and others to a rejection. Note that the domain controller has detailed information about the state and the impact of the decision it will make, but it is not the case for the orchestrator. Despite the fact that the underlying decision-making process of the controller is deterministic, the orchestrator still experiences this as stochastic, because it is unaware of the state ω of the infrastructure.

However, the acceptance probability P can be modeled by observing the outcomes of the domain controller to the previous requests. Given K observations $\{(x_1, y_1), (x_2, y_2), ..., (x_K, y_K)\}$, where $k = \{1, 2, ..., K\}$, x_k is the proposed SLO to the domain, and $y_k \in \{0, 1\}$ represents its associated response, i.e., being rejected or accepted, the acceptance probabilities for these SLA vectors $\{x_1, x_2, ..., x_K\}$ can be modelled by parameterised neural networks $\mathcal{F}$ via maximizing the following likelihood:

$$\sum_{i=1}^{K} [y_i * \log(\mathcal{F}(x_i)) + (1 - y_i) * \log(1 - \mathcal{F}(x_i))], \tag{4}$$

where $\mathcal{F}(x_i) \in [0, 1]$ is the estimated acceptance probability of the SLO x_i predicted by the neural network. Note that x is a realization of the variable s i.e., the SLO vector per domain.

C. Monotonicity

The acceptance probability based on the ground truth follows a partial order relation [3], which incorporates the notion of a stricter SLA, i.e., given a set of K SLOs $S = \{x_1, x_2, ..., x_K\}$, the acceptance probability has the following property:

$$\forall x_i, x_j \in S, P(x_i) \leq P(x_j) \quad \text{if} \quad x_i \preceq x_j, \tag{5}$$

The property indicates that a stricter SLO x_i is less likely to be accepted. Note that this is a partial instead of total order, because only a part of the vectors stand in such a relation to each other. As an example, an SLA is characterised by a (delay, throughput) tuple, the vector $x_i = (\tau_i, \theta_i)$ is stricter than the other vector $x_j = (\tau_j, \theta_j)$, i.e., $x_i \preceq x_j$ if and only if $\tau_i \leq \tau_j$ and $\theta_i \geq \theta_j$. Hence, in this case Eq. 5 states that an SLA with lower delay and higher throughput requirements is less likely to be accepted.

In order to benefit from this prior knowledge, the NN which we utilize as the risk model should also be monotonic. Many

research efforts have been carried out to incorporate monotonicity into NNs while maintaining their expressiveness [5], [6]. We introduce six approaches in Section III to learn each involved domain's risk model for SLA decomposition.

III. Approaches

In this section, the six NN-based methods are proposed, to construct a monotonic risk model for each domain. Table I summarises the characteristics of all proposed methods.

Vanilla Neural Network. A plain NN is used as the baseline model. The goal is to find a parameterized neural network $\mathcal{F}$ that minimizes the binary cross entropy loss:

$$L_{BCE} = -\frac{1}{K}\sum_{i=1}^{K}[y_i * \log(\widehat{y_i}) + (1-y_i) * \log(1-\widehat{y_i})] \quad (6)$$

where $\widehat{y_i} = \mathcal{F}(x_i)$ is the prediction by the neural network.

Regularised Neural Network. One of the sufficient conditions that guarantees the monotonicity of a NN is to have all its weights non-negative [6], [7]. To achieve this, on top of the settings of the vanilla NN, we penalize the negative weights, as an additional regularisation loss during training. Given a set of J weights $W = \{w_1, w_2, .., w_J\}$, this loss is defined as:

$$L_{Reg} = \sum_{j=1}^{J}\delta(w_j) * w_j^2,\ \delta(w) = \begin{cases} 1, & \text{if } w < 0 \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

The regularisation loss is considered together with the binary cross entropy loss in Eq. 6 during the optimization process. The total loss is then defined as:

$$L_{Total} = L_{BCE} + k * L_{Reg} \quad (8)$$

with the parameter k to balance between two loss terms. Note that the monotonicity is not strictly obeyed with this approach, as the regularisation loss is a soft constraint.

Absolute Weight Transformation (AWET). Following the incentive of a NN with non-negative weights, we consider a NN with the absolute value transformation applied to its weights before the forward computation. Thus instead of calculating $\mathbf{w} * \mathbf{x} + b$, we use $\text{abs}(\mathbf{w}) * \mathbf{x} + b$, where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, b is the bias and $\text{abs}(.)$ being the absolute value function. The weights are forced to be non-negative after the transformation, thus the optimization algorithm is allowed to update freely w during training while maintaining the non-negative weights constraint. The corresponding back-propagation is handled automatically by modern deep learning libraries, thus no additional modification is required.

Mini-batch Order Loss (MOL). Considering the pair-wise relations between samples within a mini-batch has been successfully used for training in recent years [8]. We propose the mini-batch order loss, where if the given ordered relation is not followed for all possible pairs within a mini-batch, an extra loss term is added to penalize the violations. Given a mini-batch with M samples $B = \{(x_1, y_1), (x_2, y_2), ..., (x_M, y_M)\}$, the mini-batch order loss is defined as:

$$L_{MOL} = \sum_{i}^{M}\sum_{j\neq i}^{M} S(x_i, y_i, x_j, y_j), \quad (9)$$

TABLE I: Comparison of the proposed methods

Method	strict monotonicity	weight restriction	data refinement	auxiliary loss
Vanilla	-	-	-	-
Reg.	-	✓	-	✓
AWET	✓	✓	-	-
MOL	-	-	-	✓
CSE	-	-	✓	-
PO	-	-	✓	-

$$S(x_i, y_i, x_j, y_j) = \begin{cases} \max(y_i - y_j, 0), & \text{if } x_i \preceq x_j \\ 0, & \text{otherwise.} \end{cases}$$

The purpose of the loss is to encourage the predictions for a pair of samples to follow the order relation of their inputs, if the order exists. When the predictions disobey the order, for instance, $x_i \preceq x_j$ but $y_i > y_j$, a loss occurs to correct the order relation between y_i and y_j until $y_i \leq y_j$. In conjunction with the binary cross entropy loss in Eq. 6, the total loss becomes:

$$L_{Total} = L_{BCE} + k * L_{MOL} \quad (10)$$

with the parameter k to balance between two loss terms.

Conflicting Sample Elimination (CSE). Another factor that could induce non-monotonic behavior to the model is that the training data itself violates the order relation. Therefore, preprocessing of the data becomes crucial. To address this, we propose the gradual elimination of samples conflicting the most with other samples. The algorithm is provided in Alg. 1.

Algorithm 1 Conflicting Sample Elimination

1: create table T and record the number of conflicting samples per sample
2: remove the sample with the largest number of conflicting samples
3: update table T
4: repeat step 2 and 3 until no conflicting samples exist for all samples

For example, given a set of samples $S = \{s_1 : (3, 1), s_2 : (2, 5), s_3 : (1, 2)\}$, we can see that s_1 and s_2 are conflicting samples, as $3 \succeq 2$ (here we define $\succeq$ as $>$) yet $1 < 5$, while s_2 and s_3 have no conflict because $2 \succeq 1$ and $5 > 2$. Following the same rule, s_1 and s_3 are mutual conflicting. Based on the current set S, table T is created to track the number of conflicting samples. At step 1, $T_1 = \{s_1 = 2, s_2 = 1, s_3 = 1\}$. Next, the sample that has the most conflicting samples, s_1, is removed. The set S_1 becomes $S_2 = \{s_2 : (2, 5), s_3 : (1, 2)\}$, and the table T_1 is updated to $T_2 = \{s_2 = 0, s_3 = 0\}$. The algorithm terminates when no conflicting samples exist.

Given a dataset with N samples and M-dimension features, the time complexity of step 1 in Alg. 1 is $\mathcal{O}(M * N^2)$, $\mathcal{O}(N^2)$ for the one-time sorting, $\mathcal{O}(1)$ for step 2, and $\mathcal{O}(N)$ for step 3 with a hash table. The overall time complexity of Alg. 1 is $\mathcal{O}(M * N^2 + N^2 + N * (1 + N)) = \mathcal{O}(M * N^2)$.

Probability Optimization (PO). The method proposed in [3] determines the acceptance probabilities $P = \{p_1, p_2, ..., p_K\}$ per sample by maximizing the likelihood:

$$\sum_{i=1}^{K}[y_i * \log(p_i) + (1-y_i) * \log(1-p_i)] \quad (11)$$

given the constraints in Eq. 5. The probability values are inferred from a parameter-free risk model. We leverage this

technique for pre-processing, by replacing the binary labels of the dataset with the obtained probabilities P, i.e., the dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_K, y_K)\}$ becomes $D' = \{(x_1, p_1), (x_2, p_2), ..., (x_K, p_K)\}$, where y_i is a binary value and p_i is a probability. Finally, D' is used for training.

IV. Experimental Setup

Ground Truth: To the extent of our knowledge there is no relevant data available captured on a multi-domain environment. Thus, we follow the ground truth model and data generation process in [3] to generate data for three domains.

Performance Assessment: In order to assess the performance of the proposed approaches, we repeat the following process multiple times:

Step 1. We generate K partial SLAs per each of the N domains $(\tau_{n,k}, \theta_{n,k})$ where the delay $\tau_{n,k}$ and the throughput $\theta_{n,k}$ are randomly sampled from a uniform distribution over pre-defined intervals. We use the ground truth model [3] for each partial SLA to determine the corresponding acceptance probability. Given this probability, we employ a coin toss to decide whether the partial SLA is accepted or not by the domain. Sets of partial SLAs and the corresponding binary decision is used for learning.

Step 2. We split the data in a training and a validation set, and train the NN for each domain. Training ends when the loss on the validation set no longer decreases for 100 epochs in a row.

Step 3. With these trained risk models, we decompose a given E2E SLA using exhaustive search as in [3], followed by SQP.

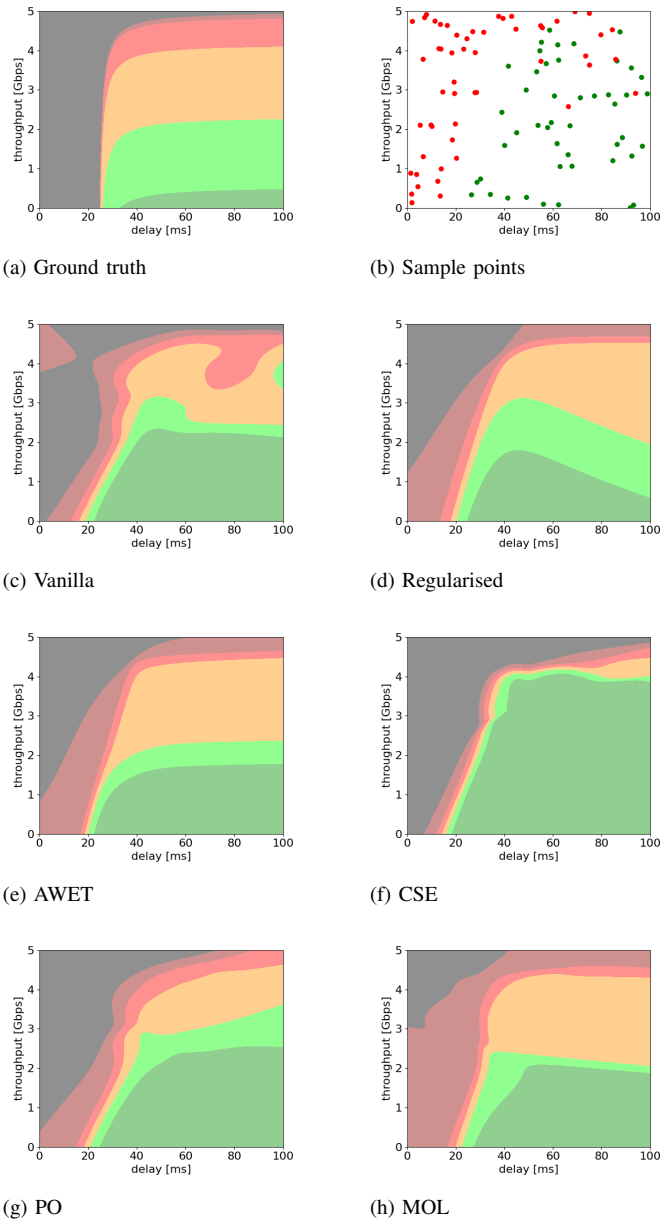
Step 4. Finally, we feed the partial SLAs to the ground truth model to determine its acceptance probability per domain and estimate the E2E acceptance probability as the product of their individual values.

We average these E2E acceptance probabilities over all independent runs and compare it to the respective probability of the optimal decomposition, i.e., based on the ground truth. The E2E SLA we employ is $(\tau_{e2e}, \theta_{e2e}) = (100\text{ms}, 0.5\text{Gbps})$.

Hereafter, we describe the configurations for training hyper-parameters and NN architectures. To have a fair comparison, methods introduced in Section III are adapted on top of the same basic configuration. All experiments are conducted on a server with Intel Core i7-10700K CPU, 32 GB of RAM.

Network Architecture. The base model is a 3-layer multi-layer perceptron (MLP), with 8 neurons each. The hyperbolic tangent (Tanh) activation function and batch normalization (BN) [9] are applied for hidden layers in the order of linear-Tanh-BN. Remark that we disable the affine transformation in BN to avoid affecting the possible monotonicity of the NN. The output layer uses sigmoid activation without appended BN. The base model is adopted by all six methods.

Hyper-parameters. For training, the learning rate is set to 0.01 and batch size to 16. The dataset is split in 80% for training and 20% for validation. The objective function to minimize for each method is described in Section III. The balance factor for regularisation loss in Eq. 8 is set to $k = 0.1$, and for the mini-batch order loss in Eq. 10 to $k = 1.0$. Both k are determined by grid search in $\{0.01, 0.1, 1.0, 10.0\}$. To reduce statistical variability, all experiments are repeated 1000 times, and average values are reported.



(a) Ground truth (b) Sample points

(c) Vanilla (d) Regularised

(e) AWET (f) CSE

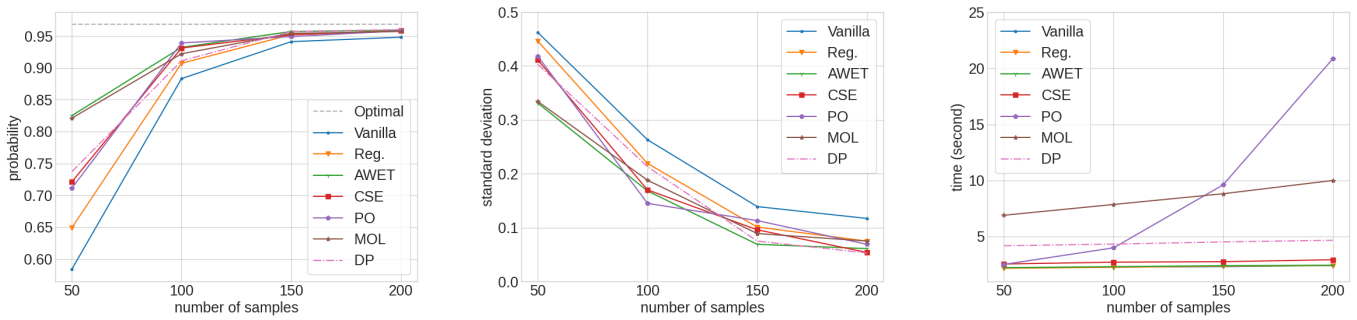
(g) PO (h) MOL

Fig. 2: Ground truth and learned risk models

V. Results

A. Risk Model Evaluation

The data (i.e. $x_i = (\tau_i, \theta_i)$, $y_i = 1$ for acceptance, $y_i = 0$ for rejection) are generated by the given ground truth model as described in Section IV (Step 1). The contour plots for all learned risk models with arbitrary 100 samples in a single domain are shown in Fig. 2{c-h}, where the dark green, light green, orange, light red and dark red region correspond to an acceptance probability greater than 0.99, 0.9, 0.5, 0.1 and 0.01, respectively. The gray region represents an acceptance probability lower than 0.01. We examine the quality of the risk models by comparing these plots to the ground truth model (Fig. 2a). Fig. 2b depicts the sampled points of our training data, where the green dots represent accepted requests while



(a) Avg. E2E acceptance probability. (b) SD of E2E acceptance probability. (c) Training time.

Fig. 3: Performance of SLA decomposition.

red dots rejected ones. From Fig. 2{d-h}, we can observe that the proposed methods with the monotonicity constraint generate smoother decision boundaries than that of the vanilla NN in Fig. 2c. Vanilla NN in Fig. 2c overfits the data with sampling error (the area with overlapping red and green dots in Fig. 2b), which leads to irregular boundaries. AWET in Fig. 2e obtains the most similar shape to the ground truth model, which is attributable to its strict monotonic constraint. CSE in Fig. 2f shows the steepest gradient at the boundaries (i.e., moving from 0.99 to less than 0.01 acceptance probability), as the overlapping red and green dots in the sampled points are removed before training.

B. SLA Decomposition

To evaluate the performance of the obtained decomposition based on the risk models, we decompose an E2E SLA of $(\tau_{e2e}, \theta_{e2e}) = (100\text{ms}, 0.5\text{Gbps})$ using the process described in Section IV considering various sample sizes (i.e., 50, 100, 150, 200). We further include the method introduced in [5] which imposes monotonicity by penalizing derivatives (DP) with respect to inputs. Fig. 3a shows the results for the E2E acceptance probability. The dashed line indicates the theoretical optimum. The vanilla NN performs poorly especially for small sample sizes, while the rest methods are improved across all sample sizes. The AWET method outperforms all models, particularly for small sample sizes, as it guarantees monotonicity. MOL method achieves similar performance to AWET, yet with notably longer training time (see Fig. 3c). However, as the sample size increases, the discrepancy in performance becomes marginal for all methods. Note that the balance factor k in both regularised NN and MOL are not optimal. We believe their performance can be improved with further parameter search. Fig. 3b shows that the standard deviation (SD) of E2E acceptance probabilities for all methods decreases inversely to the sample size. Fig. 3b in conjunction to Fig. 3a also indicates that when a model acquires larger average E2E acceptance probability has in general lower SD, which suggests higher robustness.

In Fig. 3c we can see that the training time of PO exhibits quadratic growth, due to its extra constraint handling step, as stated in [3]. The rest of the methods grow linearly over the number of samples, with the MOL having additional constant time and a larger slope for calculating pair-wise loss among samples, which is proportional to the square of batch size. DP also requires additional constant time for introducing random samples during iterations of training [5].

VI. Conclusion

In this study, we propose an approach for decomposing an end-to-end SLA associated to a network slice request to the involved domains, adopting neural network-based risk models. The approach is applicable to any two-level network slice management system. The orchestrator is unaware of the state of each domain. The risk models are built using historical data pertaining to admission control. We propose six neural network-based approaches that utilize the monotonicity prior, such that the SLA can be adequately decomposed even with a small dataset. An empirical study on a synthetic multi-domain dataset demonstrates the efficiency of our approach.

References

[1] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu, "Resource allocation for network slicing in 5g telecommunication networks: A survey of principles and models," *IEEE Network*, vol. 33, no. 6, pp. 172–179, 2019.

[2] J. Wang, J. Liu, J. Li, and N. Kato, "Artificial intelligence-assisted network slicing: Network assurance and service provisioning in 6g," *IEEE Vehicular Technology Magazine*, vol. 18, no. 1, pp. 49–58, 2023.

[3] D. De Vleeschauwer, C. Papagianni, and A. Walid, "Decomposing slas for network slicing," *IEEE Communications Letters*, vol. 25, no. 3, pp. 950–954, March 2021.

[4] M. Iannelli, M. R. Rahman, N. Choi, and L. Wang, "Applying machine learning to end-to-end slice sla decomposition," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 92–99.

[5] X. Liu, X. Han, N. Zhang, and Q. Liu, "Certified monotonic neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 427–15 438, 2020.

[6] J. Sill, "Monotonic networks," in *Advances in Neural Information Processing Systems*, M. Jordan, M. Kearns, and S. Solla, Eds., vol. 10. MIT Press, 1997.

[7] N. P. Archer and S. Wang, "Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems," *Decision Sciences*, vol. 24, no. 1, pp. 60–75, 1993.

[8] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.