iOn-Profiler: intelligent Online multi-objective VNF Profiling with Reinforcement Learning

Xenofon Vasilakos *Member, IEEE*, Shadi Moazzeni *Member, IEEE*, Anderson Bravalheri *Member, IEEE*, Pratchaya Jaisudthi *Student Member, IEEE*, Reza Nejabati *Senior Member, IEEE*, and Dimitra Simeonidou, *Fellow, IEEE*

***Abstract*—Leveraging the potential of Virtualised Network Functions (VNFs) requires a clear understanding of the link between *resource consumption* and *performance*. The current state of the art tries to do that by utilising Machine Learning (ML) and specifically Supervised Learning (SL) models for given network environments and VNF types assuming single-objective optimisation targets. Taking a different approach, *iOn-Profiler* poses a *novel VNF profiler* optimising *multi*-resource type allocation and performance objectives using adapted Reinforcement Learning (RL). Our approach can meet Key Performance Indicator (KPI) targets while minimising multi-resource type consumption and optimising the VNF output rate compared to existing single-objective solutions. Our experimental evaluation with *three real-world* VNF types over a total of *39 study scenarios* (13 per VNF), for three resource types (virtual CPU, memory, and network link capacity), verifies the accuracy of resource allocation predictions and corresponding successful profiling decisions via a benchmark comparison between our RL model and SL models. We also conduct a complementary exhaustive search-space study revealing that different resources impact performance in varying ways per VNF type, implying the *necessity* of multi-objective optimisation, individualised examination per VNF type, and adaptable online profile learning, such as with the *autonomous online learning* approach of iOn-Profiler.**

***Index Terms*—VNF Profiling, Multi-Objective optimisation, Reinforcement Learning, Resource minimisation.**

I. Introduction

THE rise of Cloud computing, Software-Defined Networking (SDN) and Network Function Virtualisation (NFV) have caused a paradigm shift from traditional networking based on specialised hardware to utilising general-purpose programmable hardware as a *resource for running Virtualised Network Functions (VNFs)*. This change has simplified the design, deployment, and management of network services, with network-based service providers offering Service Level Agreements (SLAs) to their customers that outline performance requirements *and* Key Performance Indicator (KPI) levels (e.g., throughput, packet loss, response time, processing latency, and so forth).

At the core of achieving SLA goals lies the essential *process of VNF profiling*. It involves the systematic analysis and characterisation of different VNFs within a programmable SDN environment. The primary objective is to understand each VNF's individual resource requirements, performance expectations and operational behaviour by discovering the *relationship* between resource configuration and performance. This knowledge enables service providers to decide the *(i) optimal allocation* of network and computation resources such as CPU or bandwidth, for each VNF instance, while *(ii)* ensuring *adherence* to predefined KPI thresholds after SLA goals. VNF profiling is undertaken by a "VNF profiler" and the resulting profile describes a discovered reciprocal mapping between optimised resource allocations and the KPI thresholds for the respective VNF, which enables knowing the expected performance after allocating resources and vice versa.

In the context of contemporary networks such as 5G and future 6G, attention to profiling is driven by its significance for NFV MANagement and Orchestration (MANO) systems. The latter can use VNF profiles to instantiate Network Services (NSs) by adapting optimised resource configurations. Moreover, profiles can be used to optimise the life-cycle management of running services. As an example, the 5G-VIOS [1] *common interfacility orchestration platform* leverages autonomously generated [2] profiling models to deploy and orchestrate inter-edge NSs across multiple domains and facilities by (i) *autonomously* assigning optimised resource configurations to inter-edge NSs while also (ii) exposing corresponding performance profiles.

The current work presents *iOn-Profiler*, an online VNF profiler that leverages adaptive Reinforcement Learning (RL). In summary, our most significant and novel contributions are:

1) **Online, multi-objective optimisation profiling:** We investigate RL-based adaptive VNF profiling for minimising the use of both compute and network resources, as well as finding the Optimum Output Rate (OR). The latter stands for the output rate achieved by the profiled VNF under an optimal (i.e., minimum) resource configuration that meets KPI targets.
2) **Pragmatic VNF case studies:** We consider *three pragmatic* VNFs in our experimental study, namely a virtual FireWall (vFW) and two different modes of the Snort [3] open source intrusion prevention system (Inline and Passive modes). Besides pragmatic, these VNFs span both dissimilar and similar features, allowing to assess functionality footprint on resulting profiles.
3) **Oracle exhaustive search:** We conduct an exhaustive search of resource-to-KPI combinations for all VNFs involving all resource types to establish an all-possible performance knowledge and understanding of the impact importance of different resource types on the performance of different VNF types. Then we utilise this Oracle-gained insight to carefully explore and tune the RL reward function parameters of iOn-Profiler.
4) **Extensive experimental analysis:** Overall, our evaluation study highlights that different resources impact VNF

Authors are with Smart Internet Lab, Department of Electrical & Electronic Engineering, University of Bristol, BS8 1UB, UK, Corresponding author: X. Vasilakos (e-mail: xenofon.vasilakos@bristol.ac.uk).

performance in distinct ways. Besides Oracle search, this conclusion is also established through the analysis of each VNF type's Pareto front over a total of *39 scenarios* (13 per each of 3 VNF types). Our highlight results and conclusions include:

- Multi-objective optimisation is *necessary* for proper VNF profiling.
- There is a strong requirement for studying each VNF *type* and *mode of operation individually* such as demonstrated for Snort (Passive *vs.* Inline modes).
- Online learning is *significant*, as fixed Supervised Learning (SL) models lack adaptability to dynamics.

The rest of this article is organised as follows. Section II provides the reader with the necessary background context. Section III discusses the design of iOn-Profiler. The experimental setup, resource and model configurations are described in Sec. IV. Our experimental evaluation is presented in Sec. V followed by our future work and conclusion in Sec. VI.

II. Background & Motivation in Intelligent VNF Profiling

A. Problem statement & utilising machine learning

Let I be the set of all considered *resource types* $i \in I$, and K be the set of every considered KPI type $k \in K$. Also, Let $\mathbf{x} = \{x_1, x_2, ...x_v\}$ be the decision vector of KPI threshold targets for allocating resources. Each threshold target x_k corresponds to KPI k and can get a value only from the partition set T_k defined below. Last, $\mathbf{x} \in \mathbf{X}$, where set $\mathbf{X}$ is the feasible set of decision vectors. Let the set $T_k = \{\tau_1, \tau_2, ...\tau_\omega\}$ be an partition set of considered performance thresholds for KPI type k. Last, let $f_i(\mathbf{x})$ be the allocated amount for resource type i, given the KPI threshold targets $\mathbf{x}$. We define the following *multi-objective* optimisation problem:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{X}} \left(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})\right) \\ &\text{Subject to:} \quad m_k \geq \tau_k, \forall k \end{aligned} \tag{1}$$

To consider maximum KPI thresholds as well (e.g., for packet drop rate), we adopt appropriate minimum and maximum threshold constraints τ_k^{min} and τ_k^{max} and redefine the problem constraints as: $m_k \geq \tau_k^{min}$, $\forall k$ and $m_k \leq \tau_k^{\max}$, $\forall k$.

As detailed in Sec. II-C, Machine Learning (ML) poses a dominant trend in the VNF profiling literature due to its adaptability to complex environments. Compared to other types of prominent works based on linear programming and heuristics (e.g., [4], [5]) ML solutions delve deeper into VNF-to-resource specifics, with core challenges captured *better*: First, ML can capture better network and service *dynamics*, particularly regarding 5G and 6G programmable networks due to their agility. Second, they can do so *within practical timescales* despite the *NP-hardness* [6] of the underlying optimisation problem, by *converging* towards optimised configurations involving different resources and subject to KPI targets.

Prominent examples of ML models used for profiling include Linear Regression [7], K-Nearest Neighbors Regression (KNNR) [8], Interpolation [9], Artificial Neural Networks (ANNs) [10], and Curve Fit [11]. However, it has been shown [12] that regression is not well-suited for predicting saturation regions, while SL models like ANN and KNNR, along with Interpolation, do not provide configuration trends with a monotonic rising function. In contrast, Curve Fit achieves high accuracy in predicting VNF performance but is limited in multi-objective resource optimisation. Moreover, SL models explored for service-level VNF profiling and placement may prove suitable under *static* conditions [13], however, they can significantly underperform under dynamic network conditions [14] such as in contemporary networks. Last, an important weakness of most ML works is their approach to profiling as a *single*-objective (i.e., single resource-type) optimisation problem, hence *lacking realism* as most VNFs need more than one resource types, posing a non-linear impact of allocated resource amount combinations on resulting KPIs.

B. Solution objectives

The problem statement presented above establishes the context for the current solution effort, which revolves around four primary research objectives:

Objective #1: The profiler should accommodate *multiple* resource types and KPIs, and *must* efficiently converge towards optimised VNF configurations within *practical timeframes*, despite the NP-hardness of the underlying optimisation problem.

Objective #2: Leverage *online learning* ML techniques to effectively adapt to the *dynamics* of contemporary networks.

Objective #3: Investigate the impact of *different* and *pragmatic* VNF types, with varying functionality features, on optimal resource allocation concerning specific KPI targets.

Objective #4: Conduct a comprehensive evaluation by comparing the proposed online learning solution against state of the art SL-based VNF profiler models.

In pursuit of these objectives, iOn-Profiler extends our prior work of [15] with an (i) in-depth analysis of the complex results obtained from an exhaustive search of resource-to-KPI combinations, to (ii) gain a comprehensive understanding of the relevance of resource-to-KPI and resource-to-VNF type relationships, thus enabling to (iii) fine-tune the parameters of the online learning model in iOn-Profiler. Additionally, our extension involves considering (iv) a broader set of pragmatic VNF types, encompassing different features, and exploring (v) multiple optimisation scenarios per VNF. Lastly, the article presents a (vi) meticulous experimental evaluation of state of the art SL-based VNF profiler benchmark models, including *Random Forest (RF)* and *Multi-Layer Perceptron (MLP)*. The presented research aims to advance the field of VNF profiling and contribute valuable insights into enhancing the efficiency and performance of future network architectures.

Compared to the rest of the state of the art in ML-based profiling (elaborated in Sec. II-C), iOn-profiler is designed to cover existing gaps (see Tab. I). We go *beyond single-objective* optimisation by utilising RL to better fit *real-world* applications while being *adaptable* to network dynamics. We exploit carefully designed reward functions for the multi-objective optimisation of virtual Central Processor Unit (vCPU), memory, and network Link Capacity (LC) resource allocations that can achieve desirable VNF KPIs targets such as the CPU utilisation, memory utilisation, latency and Optimum OR.

To do so, our comprehensive study considers a wide spectrum of different scalarisation weights among vCPU, memory and LC objectives, which describe the *Pareto front* of optimised resource-to-KPI combinations that we wish to approach in 39 scenarios. The Pareto front is a concept representing

the set of *non-dominated* solutions[1]. When it comes to VNF profiling the state of the art frequently ignores the Pareto front, posing a *major research gap*. Even when considered, this refers primarily to SL approaches tailored as "static" models trained for a given VNF type, under specific conditions (e.g., network structure or traffic), and therefore *cumbersome* or even impossible to generalise, if realistic at all for the highly agile and dynamic contemporary programmable networks.

C. State of the art

The state of the art discussed below is summarised and compared in Tab. I. First off, various important works [17], [18], [24] have explored *offline* profiling for VNF Service Function Chains (SFCs) with a focus on different resources. Regarding optimal VNF placement and profiling, RAVIN [21] introduces a resource-aware algorithm based on the Balanced Best Fit Decreasing (BBFD) heuristic algorithm. It enforces performance SLAs for multi-tenant NFV servers while balancing resource use, aiming to minimize server count, guarantee performance, and improve resource utilization, including the processor's Last Level Cache and Memory Bandwidth (MB). However, extensive offline profiling by exploring all possible VNF configurations such as in the aforementioned works is *time-consuming*, leading to the development of models that focus on limiting the profiling time such as in [18]. Nonetheless, and unlike our current effort in iOn-Profiler, endeavours like [18] do not encompass the concurrent consideration of pivotal KPIs such as vCPU utilization, memory utilization, latency, throughput, and packet loss. In another study by [23], researchers address the challenges of diagnosing NFV performance and introduce a metric referred to as the *Coefficient of Interference*. This metric quantifies the variations observed in latency measurements on a per-packet basis when performance diagnosis is applied against cases in which it is not employed. Last, other works, such as ORCA [16] and z-TORCH [6], have streamlined the profiling process for data collection and optimal VNF placement. However, these approaches may not consider optimal KPIs and pre-defined resource configurations.

Last, other notable contributions in the literature include the NFV-Inspector [20], an automated profiling and analysis platform, and the work of [19] utilising ML techniques such as Interpolation, Gaussian Process, ANN, and Linear Regression for predicting VNF performance.

Regarding our contribution to the field of VNF profiling, the Novel Autonomous Profiling (NAP) method [2] focuses on offline autonomous profiling by identifying the initial optimal resource configuration for each standalone VNF based on a weighted resource configuration selection approach. Furthermore, our most recent work of [22] introduces a novel autonomous *temporal* profiling technique, examining VNF behaviour across performance and resource utilisation aspects. The proposed technique automates the profiling processes, encompassing diverse resource types like computation, memory, and network resources, to yield deeper insight into VNFs resource-performance correlations.

[1]Non-dominated are best trade-off solutions, being *impossible* to further improve one objective unless compromising at least another objective.

D. The potential of Reinforcement Learning

Previous studies summarised in Tab. I demonstrate that ML model intelligence can significantly enhance accuracy and other qualitative features of VNF profiling compared to traditional methods. The current paper covers all relevant feature columns in Tab. I but that of employing SL. Instead, we employ RL and look forward to exploring its potential.

Further to our prior works, particularly NAP [2], iOn-Profiler spans both an offline and *(i)* an *online* learning phase that enables adopting to network *dynamics* at runtime, both grounded in RL. Moreover, iOn-Profiler deploys VNFs *(ii)* on the established Open Source MANO (OSM) [25] platform and suggests *(iii)* a *multi-objective* VNFs profiling strategy also grounded in RL, aiming to facilitate *Zero-touch* service and network management *automation* for dynamic networks.

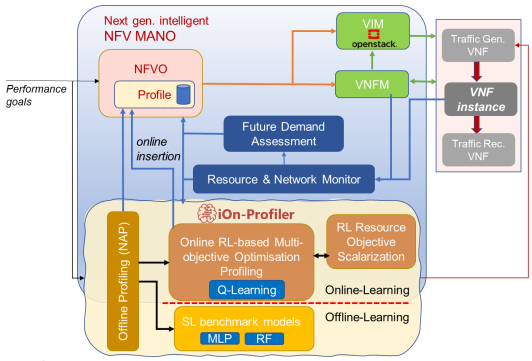


Fig. 1: iOn-Profiler and MANO integration diagram.

Within this context, iOn-Profiler marks a progression in our continuing efforts since [2] towards an *integrated* intelligent *profiling and orchestration* framework, where RL finds significance in capturing states of resource allocation and VNF node-placement through an underlying Markov decision driven by online training and live profiling feedback. The above integration can occur by fitting RL profiling agents into a more complex RL model-based orchestration scheme, or by utilising RL profilers like the iOn-Profiler in a *hierarchical* structure likewise to the one detailed in our recent work of [26].

Last, preliminary experimental evidence [x] highlight RL models' good transferability potential, enabling the transition of a model trained for profiling one known VNF type to another with reduced training costs and enhanced profiling performance.

III. iOn-Profiler Model Design

A. Integration into the next generation NFV MANO

Figure 1 illustrates the interaction between our proposed iOn-Profiler, NFV Orchestrator (NFVO), Virtualised Infrastructure Manager (VIM), and monitoring tools to provide an intelligent and autonomous NFV MANO system. The diagram not only shows interaction but also demonstrates the integration of the iOn-Profiler into next-generation intelligent NFV MANO. Through online profiling, the configuration of resources is selected and dynamically updates existing virtual network function (VNF) descriptors. As a result, the MANO system deploys a network slice with the newly defined resources. In Fig. 1, we outline iOn-Profiler's architecture.

TABLE I: State of the art summary in intelligent VNF profiling. Compared to others, iOn-Profiler "fills in" all columns corresponding to research gaps.

Ref	ML Techniques Used			Considered Resources			Considered KPIs/Metrics				Target Platform	Predicting the optimum objectives/targets			
	Supervised Learning	Unsupervised Learning	Reinforcement Learning	vCPU Cores	Memory	Link Capacity	vCPU Utilisation	Memory Utilisation	Latency	Comments		vCPU Cores	Memory	Link Capacity	Comments
Orca [16]	✔ Regression						✔		✔	Throughput, Workload	Docker				workload
Mestres et al. [17]	✔ ANN			✔						Throughput, Response time	Non-specified Hypervisor	✔			
WRVS2 [18]	✔ Regression			✔						Throughput	SOTANA				N/A
Van et al. [12]	✔ Regression, kNN, Interpolation, ANN, Curve Fit			✔		✔	✔			Packet loss	Docker	✔			Packet rate, Response time
z-torch [6]		✔		✔	✔	✔	✔	✔	✔		OpenEPC				Opt. VNF-Placement
Van et al. [19]	✔ ANN, GP, Interpolation, Regression			✔			✔			Response time, Packet rate, Workload	Docker				N/A
NFV-Inspector [20]	✔ Decision Tree			✔	✔					Arrival rate	NFV Inspector	✔	✔		Disk, Bandwidth
NAP [2]	✔ MIMO-GRNN, RF, MLP			✔	✔	✔	✔	✔	✔	Input rate	Docker	✔	✔	✔	Optimum Input rate
RAVIN [21]	N/A (not ML) - BBFD heuristic			LLC	MB					Throughput	Custom C++ platform	✔	✔		number of Servers
PDPA [22]	✔ RF, KNN DT, MLP, LR			✔	✔	✔			✔	Packet loss, Input rate	OSM	✔	✔	✔	
Ru et al. [23]	✔ Classifier			✔	✔				✔	Packet level performance, Throughput	Cisco VPP				performaqnce perturbation
iOn-Profiler	✔ RF, MLP		✔ Q-Learning	✔	✔	✔	✔	✔	✔	Output rate	MANO (OSM)	✔	✔	✔	Optimum Output rate

1) Offline Profiling: Given a series of resource availabilities and a number of KPI targets, iOn-Profiler employs the NAP method [2] to select a baseline resource configuration. Resources[2]

2) Online Multi-Objective Optimisation Profiling: After deploying the VNF with baseline resources, the iOn-Profiler employs Q-Learning (see Sec. III-B) to address possible discrepancies after moving from the staging environment where the *Offline Profiling* regression model is trained, to an *online dynamic* environment. Possible disparities are recognised when the target KPI thresholds are breached, prompting the resetting of the exploration rate and other learning parameters (see Sec. III-A2). This continuous optimisation tries to minimise resource usage without violating KPI targets and to improve allocation accuracy. Therefore, the optimisation objectives need to match the same set of resource types selected for the offline profiling, subject to the same restrictions. Each action uses the NFVO and VIM APIs to scale in/out the VNF instance and the exposed monitoring capabilities. Last, the term *online profiling* is due to the profiler (i) observing only existing network traffic without control over IR, and (ii) being used in a production environment.

[2]Resource types that can be either present or not such as smart Network Interface Cards mapped via single root I/O virtualization can be captured via profiling two corresponding VNF distinct variants. and KPIs types can be arbitrary, provided they are described by a value in a totally ordered bounded set with at least 3 elements (see Sec. IV-B, Tables III and IV). NAP is based on the concept of optimal Input Rate (IR) and OR and can be divided in 3 stages. The optimal IR is the maximum IR (in packets per second) associated with a specific resource configuration for which the system under test still respects all KPI targets, and the optimal OR is the OR associated with it.

In stage 1, NAP employs exponential ramp-up and binary search to find the optimal IR for each resource's upper/lower bounds while other resources remain at their median. Using these values, weights are calculated to measure resource influence on performance. In stage 2, NAP uses weighted random selection for applying resource configurations, measuring IR, OR, and KPIs. In stage 3, NAP trains a model to estimate minimum resource allocation based on IR and KPI targets. The method uses the NFVO and VIM to deploy the VNFs, a traffic generator and a monitoring probe at each step. Traffic generators can be employed to overcome (a) a possible lack of available real traffic datasets and (b) the need for fine-tuning the IR as required by the algorithm. We refer to the above as *offline profiling* as it needs a dataset implying the control of IR for generating arbitrary network traffic conditions, and training before the model can be used in production.

Algorithm 1: Multi-objective Q-Learning adaptation.

Input: learning rate $\alpha = 0.1$, discount factor $\gamma = 0.99$, the best steepness coefficient value (β) for each reward function, maximum number of steps (N), convergence check threshold (ε), number of steps for convergence check (N_ε) .

1 **for** $each\ objective\ o$ **do**
2 Initialise $Q_o(s, a)$ as an empty Q-table.
3 **end**
4 **for** $each\ episode\ t$ **do**
5 Initialise present state s vector;
6 Initialise circular buffer Δs with N_ε slots;
7 **for** $n \leftarrow 0$ *to* N **do**
8 **if** $\mathrm{rand}(0, 1) < \epsilon$ *-greedy* **then**
9 Action $a \leftarrow \mathrm{rand}\ A(s)$;
10 **else**
11 Action $a \leftarrow$ Call Algorithm 2 for state s;
12 **end**
13 Take action a and observe the next state s'
14 Calculate reward ($\boldsymbol{R}_o$) of each resource in s' through equation (2).
15 Call Algorithm 2 to find a' based on s' that gives the maximum Scalarised Q-value
16 **for** $each\ objective\ o$ **do**
17 $Q_o(s, a) \leftarrow Q_o(s, a) + \alpha\left(\boldsymbol{R}_o + \boldsymbol{\gamma} Q_o(s', a') - Q_o(s, a)\right)$;
18 **end**
19 Ask the NFVO to scale in or out the VNF based on s'
20 Find the OR and record the corresponding state
21 Insert $\|s - s'\|$ in Δs;
22 $s \leftarrow s'$;
23 **if** $n > N_\varepsilon$ ***and*** $\max \Delta s < \varepsilon$ **then**
24 The algorithm has converged;
25 **stop**
26 **end**
27 **end**
28 **end**

B. Multi-objective reinforcement learning model adaptation

Algorithm 1 describes our multi-objective RL approach to optimising resource allocation for a given type of VNF. This approach is aimed at addressing a Markov decision process by dynamically constructing Q-tables (Q_o) for each optimisation objective (o) that stores the estimated discounted sum of future rewards for each possible action (a) at a given state (s). The Q-tables gradually converge by exploring the action space and performing updates based on the recursive Bellman equation (shown in line 17) Our model considers the following definitions for state (a), action (a) and reward (R_o):

1) State: A vector that encompasses allocated resources (e.g., vCPU cores number) in addition to the measured KPIs (e.g., vCPU utilisation) and OR.

2) Action: The set of feasible actions encompasses increasing, decreasing, or preserving resource assignments. These actions induce shifts between various states of allocation (e.g., incrementing/decrementing the number of vCPU cores). In terms of action choice, we employ a scalarized ϵ-greedy algorithm, which facilitates the selection of actions that optimise individual rewards for each resource category by selecting the action with the highest reward with probability $1-\epsilon$.

Algorithm 2: Scalarised Greedy Action Selection.

Input: $w_o \leftarrow$ The weight of each objective, $s \leftarrow$ observed state.
1 $SQlist \leftarrow \{\}$;
2 **for** *each action* $a \in A$ **do**
3 $\quad v \leftarrow \hat{Q}(s,a) = \{Q_1(s,a), Q_2(s,a), ..., Q_m(s,a)\}$;
4 $\quad \widehat{SQ}_{linear}(s,a) \leftarrow f(v,w)$;
5 $\quad$ Append $\widehat{SQ}_{linear}(s,a)$ to SQlist;
6 **end**
7 **return** $\underset{a'}{\text{argmax}}\ SQlist$

3) Reward Function: To find the reward function for each VNF type, we consider and optimise the parameters of the following reward function model. For each resource type (i.e., objective), we use the *zedoid* (i.e., a reverse sigmoid) general formula of $f(x) = \frac{1}{1+e^{x}}$. Zedoid function allows to adaptively/gradually yield reward values that *decay with increased resource* allocations and vice versa. Therefore, the rewards promote a more cost-efficient use of resources. We

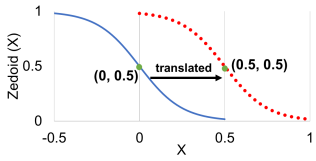


Fig. 2: Translation of *zedoid* function by 0.5 units to yield rewards only for positive resource allocation over x-axis.

adopt an appropriately parametrised (discussed in Sec. IV-D) version of the zedoid function depicted in Fig. 2. The adopted zedoid is shifted by 0.5 units. This is a desired transposition of the zedoid curve so that reward values reflect meaningful (i.e., positive) resource allocations over the x-axis. It is worth noting that in Fig. 2, the blue solid graph curve and the translated red dotted curve follow the $\frac{1}{1+e^{8x}}$ and $\frac{1}{1+e^{8(x-0.5)}}$ formulas, respectively. We also impose a penalty for constraint violation (including KPI targets) by mapping the computed value to 0. The general formula of the adopted reward function for each resource type is defined in (2):

$$R_o = \begin{cases} \dfrac{1}{1+e^{\beta(\hat{o}-0.5)}}, & \text{constraints satisfied} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $\hat{o}$ is allocated resource, i.e., the number of allocated vCPU cores, the amount of allocated memory or LC; and β is the steepness coefficient of the resource reward function that defines a desired curve steepness best fitting a resource type's adaptability to allocation changes. We return and optimise the selection of β in Sec. IV-D. We extend the scalarisation function from single to multiple objective calculations, as in Algorithm 2. For each action at line 2 - 3, Q values from all objectives are put in a vector as (3). Note that m refers to each optimisation objective. This vector and a weight vector $w = (w_1, w_2, \ldots, w_m)$ are applied to the scalarisation function $f(v,w)$ to calculate the scalarised Q-value (SQ) according to (4). The sum of all weights must be 1. At line 5, SQ is appended to the $SQlist$. Finally, at line 7 the algorithm returns the action a' corresponding to the highest SQ.

$$\hat{Q}(s,a) = \{Q_1(s,a), Q_2(s,a), ..., Q_m(s,a)\} \quad (3)$$

$$\widehat{SQ}_{linear}(s,a) = f(v,w) = \sum_{o=1}^{m} w_o\, Q_o(s,a), \text{where} \sum_{o=1}^{m} w_o = 1 \quad (4)$$

C. Solution complexity and practical costs

The execution (*time* and *memory*) complexity of iOn-Profiler is defined by the interaction between actions and the state space of the underlying Q-learning process. Each state includes the allocated resource values, KPI thresholds, parameters referring to the input passed to the VNF (e.g., input requests traffic), and last, the observed KPI measurements. As such, the state space complexity is defined by the count of (a) the resource types considered, (b) the input types, and (c) the targeted KPIs. Given the former, the asymptotic execution complexity is also a function of (i) the *granularity* of possible resource assignment levels per resource; (ii) the number of resources; (iii) the measurement granularity per KPIs; and last the (iv) possible input levels per each input type.

Further to the problem definition (Sec. II-A), let $\lambda = |I|$ be the number of resource types considered and $\kappa = |K|$ the count of the different KPIs targeted. Let Ψ be the set of VNF input categories, with $\zeta = |\Psi|$. Also, let each i be assigned values in $\{\iota_1, \iota_2, ...\iota_\rho\}$. Note that the granularity of the latter counts ρ feasible resource level allocation options for each i. For coherence, we classify measurements for each k into the immediately preceding class within set T_k, thus counting ω measurement options (Recall T_k from Sec. II-A). Finally, let set $U_\psi = \{\upsilon_1, \upsilon_2, ...\upsilon_\eta\}$ define a partition of possible VNF input levels per type $\psi \in \Psi$, thus counting η levels.

The above gives an *an asymptotic upper bound* for the space complexity and time needed to explore the whole space of $O(\rho^\lambda) \times O(\omega^\kappa) \times O(\eta^\zeta)$, which falls within the *exponential* complexity class $O(c^n)|c > 1$. *Practical implementations* of Q-learning solutions set thresholds for action steps, hence reducing memory and time costs significantly. Another aspect of time costs refers to the adopted learning rate in RL.

The demonstrative implementation setup considered in the current paper assumes *three* types of resources; *one* input type: input traffic to the VNF; and *four* target KPIs with one threshold target each (see Tab. IV). As such, the state space is represented by a *vector* of *nine* elements, including the allocated vCPU cores, memory, and output LC, the four KPI measurements, and input traffic value, and the computed scalarised Q value. Given the adopted resource configuration values in Tab. III and allocation steps in Sec. IV-B, the implied memory needs include nine 16-bit float numbers (i.e., 144 bits) per state[3]. Given the resource levels assumed in Sec. IV-B, and information in Tab. III and Tab. IV, there are $6 \times 6 \times 8 = 288$ resource level combinations, and $2^4 = 16$ KPI alignment/violation combinations. As a result, there are $288 \times 16 = 4608$ states in the state space implying an 81 KB memory need. This is marginally lower than typical first-level

[3]Note, that the previous numbers can be significantly compressed by using bitmaps of deltas rather than 16-bit float or short numbers. Caching recently computed Q values can also save a lot of exploration/exploitation.

CPU data cache sizes (e.g., 16-128KB), thus allowing us to benefit from fast computations. Besides the memory cost, the mean time cost of each training episode in our experiments is 738 steps, defined by the convergence of the Q value.

IV. Experimental Setup

Fig. 3 depicts our profiling experimental setup assuming Snort or vFW as the VNF instance. It shows the connection between the profiled VNF on the one hand, and the traffic generator and server end-point machines on the other. The two end-point machines have two vCPU cores, 2 GB of memory and 10 GB of storage. For simplicity, we employ iPerf as a traffic generator with UDP packets, noting that active data collection is more suitable for RL than static datasets.

We employ the Prometheus and Node exporter monitoring tools to gather the following metrics: vCPU utilisation, memory utilisation, and ingress and egress traffic rates to and from the VNF, respectively. Additionally, we calculated the mean Round Trip Time (RTT) using the ping utility. In addition, the duration for the offline profiling was set to 48 hours for each VNF model. The software tools and frameworks used in this study are outlined in Tab. II.

A. VNF type scenarios

We evaluate the performance of our proposed method using three different types of VNFs as our experimental scenarios. These VNFs cover a range of scenarios and demonstrate varying sensitivities to different resources. For example, the performance of the copying VNFs may be more impacted by memory utilisation, while the intercepting VNFs may be more impacted by vCPU utilisation.

TABLE II: Software frameworks and tools.

Software type	Functionality provided	Version
OSM	Orchestrate & run VNF instances	8.0.4-1
Snort	Inline & Passive mode	2.9.17
iPerf 3	Generate traffic & measure the bandwidth	3.1.3
Ubuntu	The OS on each VM	20.04
Prometheus	Monitor the performance metrics	2.25.0
Node exporter	Gather information from Linux services	1.1.2

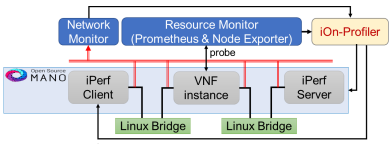


Fig. 3: Experiment setup

1) **Snort (Inline mode):** The Snort VNF operates as a traffic gateway between network segments and inspects all incoming packets before forwarding them to the destination. This mode slows down traffic transmission and may block suspicious packets.
2) **Snort (Passive mode):** The Passive mode Snort VNF operates outside of the direct traffic path and copies incoming traffic to detect suspicious activity. This mode raises a different set of resource needs compared to the Inline mode, as shown by our evaluation results.
3) **Virtual Firewall (vFW):** Allows packets to pass only through specified ports towards the destination server.

B. Resource and KPI targets configuration

We consider vCPU cores, memory and LC as our profiled resources. Tab. III shows the upper and lower bounds for their configuration values, chosen in accordance with our experimental environment and the specs of the considered VNF types. The iPerf traffic generator client transmits UDP packets with an initial traffic rate of 50 Mbps to the destination iPerf server. The traffic rate gets gradually increased, and the assumed KPI thresholds are specified in Tab. IV.

TABLE III: Resource configuration.

Configuration	values
Minimum and Maximum value of Resources:	
number of vCPU cores	0.6 - 1.8 (30% to 90% of 2 cores)
Memory (MB)	1000 - 1600
Link Capacity (Mbps)	400 - 800

TABLE IV: KPI targets that the VNF under profiling should meet.

KPI targets	values
vCPU Utilisation (%)	95±5
Memory Utilisation (%)	$\leqslant$ 98
Latency (ms)	2.5±5

C. State transition actions and training episodes

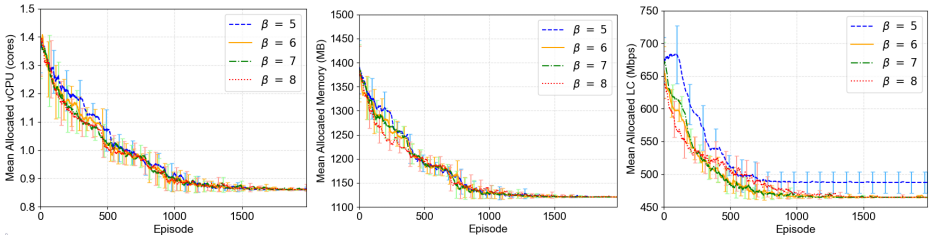
In terms of actions, vCPU is increased/decreased by 0.2 cores, memory by 100 MB, and LC by 50 Mbps, which aligns quantisation of state space and implementation necessities after the complexity analysis in Sec. III-C. For the scalarised ϵ-greedy algorithm, we adopt a decay factor $\epsilon = 0.9999$, with minimum exploration rate 0.1, learning rate $\alpha = 0.1$, and discount factor $\gamma = 0.99$. Training is organised in episodes encompassing action steps until either a maximum number of steps is reached or the minimum resources are found. Given this setup, a total of 2000 episodes was assumed, encompassing a mean number of 738 steps per episode.

D. Rewards configuration

We conducted an experiment-based parameter tuning of our reward functions to optimally adjust parameters, such as the steepness coefficient β, to the unique requirements and characteristics of the three different VNF types and to the impact of the three different resource types on profiling performance. This resulted in 9 individual reward parameterisations. To achieve this, we analysed each resource type for each VNF type in isolation. This involved using a fixed resource allocation value for the other two resource types to speed up the process. The mean values of the other two resource types, which yield optimal allocation of the investigated resource type in a controlled environment (i.e. minimum resource usage for maximum OR), were used as the fixed values.

1) Snort (Inline mode):

- **vCPU** ($R_{cpu}; \beta = 8$)**.** Regarding vCPU, Fig. 4(a) shows how the model learns to adapt vCPU values from 0.6 to 1.8 cores, assuming fixed mean memory and link capacities equal to 1300 MB and 600 Mbps, respectively. Reward R_{cpu} (for all different β configurations) exhibits a similar performance, and convergence point after episode



(a) (b) (c)

Fig. 4: Configuring reward function parameters for Snort with Inline mode.

1000, resulting in 0.88 vCPU cores. However, the best reward function based on the smallest confidence intervals is R_{cpu} for β = 8.

- **Memory** ($R_{mem}; \beta = 7$)**.** Regarding memory, Graph (b) of Fig. 4, it considers fixed mean values of 1.2 vCPU cores and 600 Mbps of LC. R_{mem} with a different β parameter value settings allocates converges to approximately 1123 MB after approximately 1200 episodes. The best reward function based on the smallest confidence intervals is R_{mem} for β = 7.
- **Link Capacity** ($R_{lc}; \beta = 7$)**.** We study LC for 1.2 vCPU cores and 1300 MB of memory. As portrayed in graph 4(c), the best reward option is R_{lc} with steepness coefficient $\beta = 7$, which yields an LC approximately equal to 475 Mbps whereas converging to this optimal final state faster than alternatives after episode 1250.

2) Snort (Passive mode):

- **vCPU** ($R_{cpu}; \beta = 8$)**.** Fig. 5a shows the results of an experiment in which the model is trained to adapt the number of vCPU cores from 0.6 to 1.8 whereas keeping the values of memory and LC fixed at 1300 MB and 600 Mbps, respectively. The R_{cpu} decreases from 1.44 to about 0.88 at episode 1500 in all the alternatives of the steepness coefficient values (β). However, the best reward option is R_{cpu}, $\beta = 8$ as it converges to the final state faster than alternatives after approximately 1400 episodes.
- **Memory** ($R_{mem}; \beta = 7$)**.** In Fig. 5(b), R_{mem} graph scales down from 1425 MB to 1147 MB assuming fixed mean values of 1.2 vCPU cores and 600 Mbps of LC after approximately 1600 episodes. The most effective reward function for minimising memory is when $\beta = 7$ based on the smallest convergence time.
- **Link Capacity** ($R_{lc}; \beta = 8$)**.** Fig. 5(c) shows the mean allocated LC for 1.2 vCPU cores and 1300 MB of memory. However, the best reward function based on minimum LC is for $\beta = 9$ where LC can be reduced from 689 Mbps to approximately 480 Mbps at episode 1525.

3) Virtual FireWall:

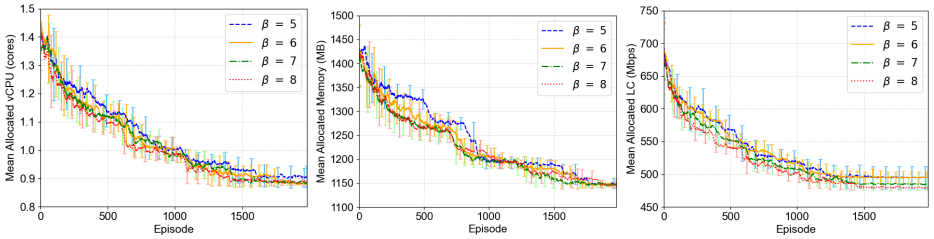
- **vCPU** ($R_{cpu}; \beta = 7$)**.** Fig. 6(a) demonstrates an experiment where the model was trained to vary vCPU values from 0.6 to 1.8 cores while maintaining memory and LC at 1300 MB and 600 Mbps, respectively. Using R_{cpu}, it can be seen that the vCPU cores get reduced from 1.40 to approximately 0.87 at episode 1450. However, we find the minimum vCPU cores faster using $\beta = 7$.
- **Memory** ($R_{mem}; \beta = 7$)**.** For R_{mem} in Fig. 6(b), the system using the R_{mem} reward function, tries to make adjustments to decrease the memory from 1430 to 1140 MB when the vcpu cores are 1.2 and LC is 600 Mbps. However, the best reward function based on the minimum memory is R_{mem} for $\beta = 7$ at episode 1600.
- **Link Capacity** ($R_{lc}; \beta = 9$)**.** As shown in Fig. 6(c), we use R_{lc} to find the minimum LC where vCPU cores are 1.2 cores and memory is 1300 MB. However, $\beta = 9$ can significantly reduce LC from 686 Mbps to 482 Mbps.

V. Performance Study

We conduct a comprehensive search in Sec. V-A to discover an "Oracle" model of optimal profiles in a simulation environment. These optimal solutions set the ultimate performance targets for our RL model. Moreover, a practical assessment of our approach requires a comparison against intelligent models thus we train SL models and compare their performance against online learning over a *dynamic environment* with growing dataset size, so as to draw adaptability conclusions.

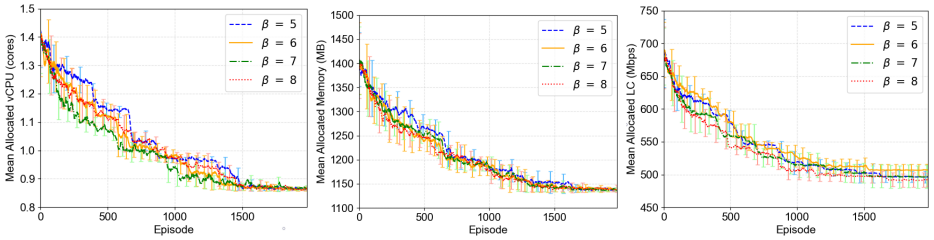
A. Oracle resource allocation (exhaustive search study)

The results presented in Fig. 7, 8 and 9 correspond to each VNF type, namely Snort for Inline mode, Snort for Passive mode and vFW, respectively. Each figure contains 5 graphs that portray performance after an *exhaustive* exploration of resource allocation combinations towards identifying an *optimal trade-off* combining a minimum of resources for optimal performance in terms of OR. All performance measurements are based on the mean values of at least *30 recorded instances* from a dataset attained during the offline profiling stage, alongside corresponding *95%* confidence intervals. Graphs (a) and (b) in Figures 7, 8 and 9 show the mean OR against the number of allocated vCPU cores and LC, respectively, for different allocated memory levels mapped to each curve in the graphs per each VNF type. Their purpose is to *pinpoint a minimum* of resource allocation on the x-axis for which the OR on the y-axis converges to a maximum mean value. Specifically, Graph (a) in each figure above illustrates the impact of vCPU cores on OR with a fixed LC of 600 Mbps, while Graph (b) shows the impact of LC with a fixed allocation of 1.2 cores for vCPUs. Fixing these values serves to focus on the direct relationship between pairs of values. Note that fixed values are carefully selected to accommodate Optimum ORs after preliminary test runs. Graphs (c) and (d) plot mean OR (orange curves) compared to consistently increasing LC levels (blue curves). The y-axes show bit-rates against increasing LC levels grouped by increasing vCPU cores or memory for (c)



(a) (b) (c)

Fig. 5: Configuring reward function parameters for Snort with Passive mode.



(a) (b) (c)

Fig. 6: Configuring reward function parameters for vFW.

and (d), respectively. If these two curves *identify*, then the LC is *best-utilised*, with the best resource combinations achieved at optimal (i.e., maximised) OR levels. As with (a) and (b), we keep memory fixed at 1300 MB for (c) and vCPU at 1.2 cores for (d). Last, Graph (e) in each figure shows on the x-axis increasing memory levels grouped by incrementally increasing vCPU cores: 0.6, 0.8, ..., 1.8, given fixed 600 Mbps.

1) Snort (Inline mode): The graphs of Fig. 7(a) and Fig. 7(b) show three curves corresponding to 1300 MB, 1500 MB and 1600 MB memory levels. We observe that the OR grows with the number of vCPU cores in the range of *0.6 – 1.4* cores regardless of allocated memory in Fig. 7(a), excluding the case of 1.0 - 1.2 vCPU for the 1500 MB and 1600 MB memory curves due to outliers as denoted by confidence intervals. The OR also increases with LC in the Graph 7(b) for all memory curves. By comparing the different memory curves, increased memory results in a higher OR. Based on the above, we can conclude that the total allocation of *all resource types* collectively affects the OR. Also, *OR converges* to a maximum of ˜550 Mbps after $vCPU = 1.4$. Another important conclusion from Graph (b) (also backed by conclusions below after Graph (d)) is that the OR for a memory of less than 1500 MB ceases to increase and is, thus, *sub-optimal*. At the same time, an increased memory allocation at 1600 MB does not increase OR further. Regarding Graphs (c) and (d) of Fig. 7 the optimal utilisation of LC can be achieved with *minimum vCPU 1.4*, as OR for increasing LC in Graph (c) slowly converges and *finally identifies with LC* at a minimum (i.e., optimal) allocation of vCPU 1.4. Note that this is consistent with the observation from Graph 7(a) (see above). The *best LC utilisation can be achieved with a minimum of memory* (1500 MB), as OR in Graph (d) for increasing LC identifies with LC for a minimum (i.e., optimal) memory level of 1500. For completeness, we note that lower memory levels like for 1100 MB show a linear (but not identifying) trend between OR and LC curves, yet with large confidence intervals. Last, Graph (e) of Fig. 7 leads to the conclusion that OR (orange curve) generally *increases with vCPU* until before $vCPU = 1.4$ irrespective of some large confidence interval values, *and then converges* for vCPU≥1.4. This is consistent with the observation from Graph 7(a), and with the conclusion from Graph 7(c).

2) Snort (Passive mode): Likewise to Snort for Inline mode, the conclusions for each graph of Fig. 8 are as follows. Regarding Graph (a), increasing vCPU cores causes a higher OR. However, the OR converges and remains at around 525 Mbps in the range of 1.0 – 1.8 vCPU cores. This holds for all memory level curves, for which LC results strongly identify. Graph (b), on the other hand, shows that the OR increases in an almost linear function with LC at 1.2 vCPU cores at all memory sizes. In addition, Even though we added more memory across all vCPU cores, as also shown in Graph (e), the OR at each vCPU core remained the same. Therefore we conclude that *memory does not impact OR*. This is due to this VNF type's different nature compared to Snort (Inline mode), with the latter needing memory resources to inspect packets before forwarding them. The OR in Graph (c) is similar to LC in the range of 1.0-1.8 of vCPU cores, while in Graph (d) OR changes along with LC across the memory range. We conclude that the vCPU cores and LC affect OR, but memory does *not*.

3) virtual FireWall: The graphs of Fig. 9 for the case of vFW are similar to the ones for Snort (Passive mode), where the OR depends on the LC across the vCPU core and memory range. Nevertheless, for vCPU equal to 0.6 in Graphs (a) and (c), the OR also seems to depend on the vCPU core. Moreover, Graphs (a), (b), and (e) show that the output value does not change even when the memory is increased. And last, the LC obviously affects the OR, as shown in Graphs (b) and (d).

B. Online learning profiling performance

We assess iOn-Profiler's Q-Learning adaptation of Algorithm 1 in a dynamic environment where the dataset size

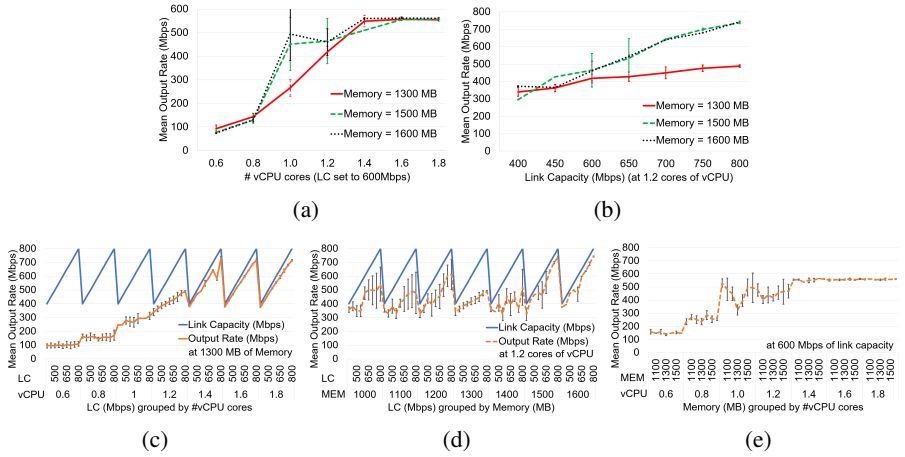


Fig. 7: Oracle resource allocation using Snort (Inline mode).

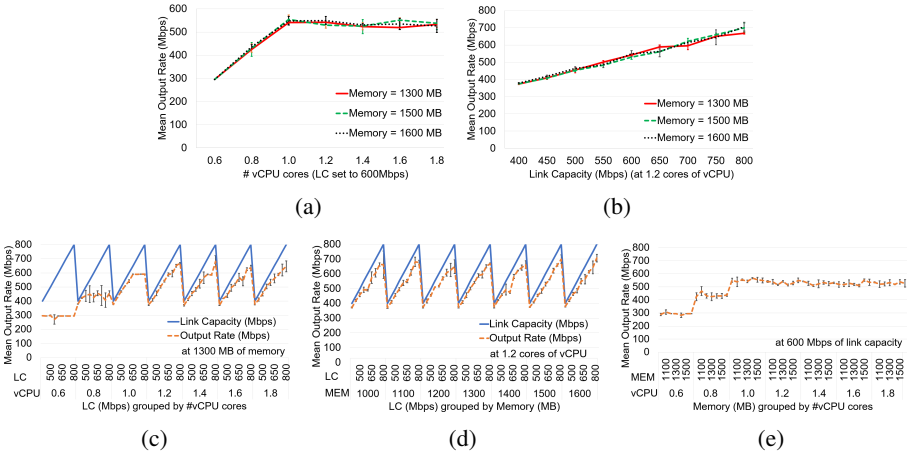


Fig. 8: Oracle resource allocation using Snort (Passive mode).

grows at run time. We compare the predicted resources to those obtained in a static environment with a static dataset for each training episode, as shown in Fig. 10 and Fig. 11. We calculate the percentage error in resource allocation compared to the optimal allocation, along with 95% confidence intervals for each resource type and the reported results refer to a scenario of[4] *equal resource* weights w(⅓, ⅓, ⅓) and tuned parameters β for each resource in a static environment.

1) Setup and training of SL benchmarks: For the MLP and RF benchmarks, we forecast the resources required periodically at landmark episodes (where RL performance is recorded and depicted in Figures 10-12) by training the models on the available dataset collected up to that episode. Then, we split the dataset into a 90:10 ratio for the training and test sets, and normalised it using min-max feature scaling. This approach allows us to conduct a fair comparison between RL and the benchmarks over the same training data.

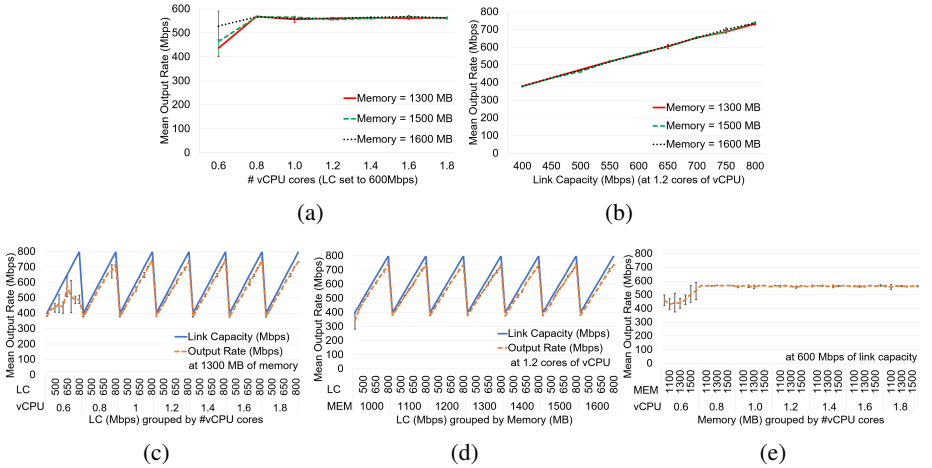
The input variables for the SL predictions include vCPU and memory utilisation, latency, and Optimum OR, while output variables include the number of vCPU cores, memory, and LC. The number of trees in the RF is set to 500, 500, and 800 for Snort (Inline mode), Snort (Passive mode) and vFW, respectively. The MLP parameters are described in Tab. V.

[4]We elaborate on this in Sec. V-C, where scenario w(⅓, ⅓, ⅓) is only one out of 13 for each of the 3 VNF types outlined in Tables VIII, VI and VII.

TABLE V: Parameters of the MLP Model

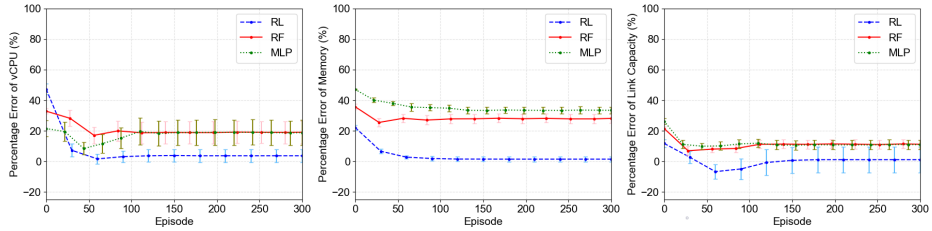
Parameter	Snort (Inline)	Snort (Passive)	vFW
Number of neurons in Input Layer	4	4	4
Number of neurons in Output Layer	3	3	3
Number of neurons in 1st Hidden Layer	512	256	128
Number of neurons in 2nd Hidden Layer	256	128	128
Number of neurons in 3rd Hidden Layer	256	128	128
Activation Function in Hidden Layer	selu	selu	selu
Activation Function in the Output Layer	sigmoid	sigmoid	sigmoid
Epoch	500	500	500
Batch size	16	16	16
optimiser	Adam	Adam	Adam
Learning rate	1e-4	1e-4	1e-4

2) Snort (Inline mode): Graphs (a), (b), and (c) of Fig. 10 show the resource allocation percentage error for Snort with Inline mode. According to (a) and (b), RL has less vCPU and memory percentage error than MLP and RF. As for Fig. 10(c), MLP and RF do not significantly reduce LC whereas the RL gives a lower percentage error. We can infer from the data above that RL can provide a lower percentage of prediction resource error than MLP and RF. The underlying reason is that RL learns to reduce resource consumption from past events. In contrast, the resource allocation percentage error of MLP and RF are high because they use a static trained model that makes them unable to adapt to reduce resource consumption.



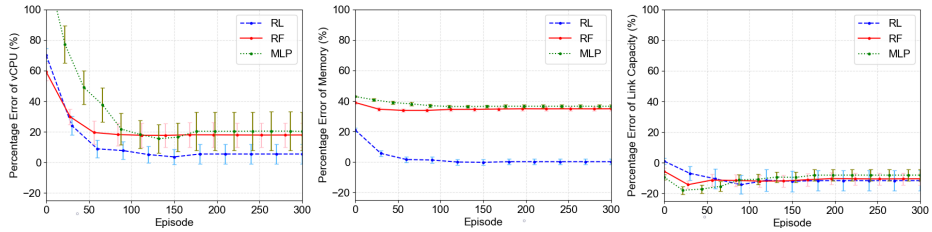
(a) (b) (c) (d) (e)

Fig. 9: Oracle resource allocation using vFW.



(a) (b) (c)

Fig. 10: Percentage error of resource allocation predictions by MLP, RF and RL regarding Snort (Inline mode).



(a) (b) (c)

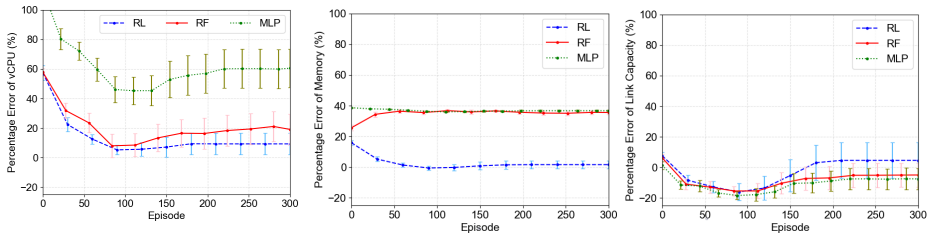
Fig. 11: Percentage error of resource allocation predictions by MLP, RF and RL regarding Snort (Passive mode).

3) Snort (Passive mode): According to Graph (a) of Fig. 11, RL produces a lower percentage error regarding vCPU cores than MLP and RF. In terms of Memory, Graph (b) shows that RL yields *no* error contrary to MLP and RF. In Graph (c), the LC percentage error is negative and quite similar for RL, MLP, and RF. Therefore, RL is more accurate for Snort with Passive mode as it yields lower error percentages than MLP and RF, notably for vCPU and memory. The analysis presented in Figure 12 focuses on the performance of the vFW VNF. In general, our online RL model exhibits notably superior capabilities for predicting resource allocation compared to the benchmarks across all resources. When examining Graphs (a), (b), and (c) after 150-175 episodes[5], the RL model demonstrates mean percentage errors of 9%, 2%, and 5% for vCPU cores, memory, and LC respectively. In contrast, the MLP and RF models yield errors of 52% and 18% for vCPU cores, respectively, and produce a 37% error for memory, and -5% and -6% of error, respectively, for LC. Noteworthy, all models achieve an error close to 0% for LC, posing a significant finding considering the substantial impact of LC on the performance of the vFW. However, negative errors by the benchmarks indicate under-provisioning predictions compared to the required LC. The over-provisioning predictions made by the RL model are preferred over the under-provisioning exhibited by the benchmarks, as the latter results in sub-optimal OR performance of the vFW.

[5]This range poses an approximate performance convergence milestone across all graphs and models.

C. Resource optimisation scenarios

We examine the impact of resource objectives on performance as a result of the resource type importance on the optimisation problem. The latter is captured via weighted parameters in the scalarised Q-Learning equation of formula (4). We investigate *39 scenarios* (13 per VNF type) with different RL resource allocation objective weights, with our findings presented in Tables VI, VII, and VIII for each type of VNF.



(a) (b) (c)

Fig. 12: Percentage error of resource allocation predictions by MLP, RF and RL regarding the vFW VNF.

1) Snort (Inline mode): Our findings in Tab. VI show that all weight configurations can reduce vCPU core usage to around 40%. Specifically, setting the weight of vCPU to one ($w(1,0,0)$) leads to a 40% reduction in vCPU usage, but also results in an 80% reduction in memory usage and a low link utilisation of 34.38% (as expressed by $^{OR}/_{LC}$). On the other hand, weight configurations such as $w(1/2,1/2,0)$ and $w(1/2,0,1/2)$ increase vCPU usage, reduce memory usage, and increase link utilisation to almost 80% respectively. Our analysis reveals that the weight of LC has a higher impact on vCPU usage than the weight of memory reward. Furthermore, the weight of vCPU and the weight of LC do not affect memory usage. The steady-state LC utilisation for $w(0,0,1)$, $w(0,1/2,1/2)$, and $w(1/2,0,1/2)$ is 92.73, 91.70, and 79.54 respectively. If the weight of vCPU is increased, the OR/LC decreases significantly, while the weight of memory has no significant effect on the OR/LC. Finally, compared to schemes with high resource weighting or equal weighting, $w(1/2,1/2,0)$, which only weights vCPU and memory, does not increase link utilisation. As a result, considering the weight of LC is critical in enhancing LC utilisation.

TABLE VI: Mean steady-state output for Snort (Inline mode) in 13 resource-scalarisation scenarios.

		vCPU	MEM	LC	vCPU (2 cores)	MEM (1.6 GB)	OR/LC (%)
Single Resource Weighted	vCPU	1			40%	100.00%	34.38
	MEM		1		90%	68.81%	91.62
	LC			1	90%	100.00%	92.73
Two Resource Weighted	vCPU & MEM	1/2	1/2		44%	69.94%	41.25
	MEM & LC		1/2	1/2	90%	69.69%	91.70
	vCPU & LC	1/2		1/2	50%	100.00%	79.54
High Resource Weighted	vCPU	3/4	1/8	1/8	41%	68.00%	59.82
		1/2	1/4	1/4	41%	67.75%	57.87
	MEM	1/8	3/4	1/8	42%	68.38%	58.97
		1/4	1/2	1/4	41%	68.25%	59.95
	LC	1/8	1/8	3/4	42%	68.00%	60.78
		1/4	1/4	1/2	41%	67.62%	60.00
Equal Weights	vCPU & MEM & LC	1/3	1/3	1/3	41%	66.31%	52.20

2) Snort (Passive mode): The case of only vCPU $w(1,0,0)$ in Tab. VII minimises demand for vCPU cores to 40%. vCPU usage can be reduced to 44% and 42% also in the case of $w(1/2,1/2,0)$ and $w(1/2,0,1/2)$, respectively. Memory weight has a slight impact on allocated vCPU cores. Memory and LC weights have a minor impact on allocating vCPU cores. The weight of vCPU has more impact on memory allocation than that of LC. The weight of vCPU has a greater influence on the LC utilisation than the memory weight. In conclusion, Snort Passive mode with $w(1/3,1/3,1/3)$ can effectively reduce resource usage while achieving a high link utilisation OR/LC.

3) Virtual FireWall: The need for vCPU cores is reduced to 40% for the case of $w(1,0,0)$. In the cases of $w(1/2,1/2,0)$ and $w(1/2,0,1/2)$, vCPU usage can be decreased to 42% and 40%, respectively. In this case, LC and memory weights have no impact on the allocation of vCPU cores. Memory allocation for the cases of $w(0,1,0)$, $w(1/2,1/2,0)$, and $w(0,1/2,1/2)$ is reduced to 55%, 56%, and 56%, respectively. In this case, the weight of vCPU and LC does not impact memory. For weight combinations $w(0,0,1)$, $w(0,1/2,1/2)$, and $w(1/2,0,1/2)$, link utilisation OR/LC in steady state is 94.45%, 94.64%, and 88.84%, respectively. Consequently, link utilisation is influenced more by the weight of vCPU than by the weight of memory. Regarding all high resource-weighted cases, steady-state vCPU and memory were about 39% and 55%, respectively. However, for $w(1/2,1/4,1/4)$ we observe the highest link utilisation OR/LC, almost 90%, which is about 9% higher than the case of the high weight of the LC (around 81%). To conclude, vFW with $w(1/2,1/4,1/4)$ can cut down on resource consumption while offering high OR/LC.

TABLE VII: Mean steady-state output for Snort (Passive mode) in 13 resource-scalarisation scenarios.

		vCPU	MEM	LC	vCPU (2 cores)	MEM (1.6 GB)	OR/LC (%)
Single Resource Weighted	vCPU	1			40%	100.00%	61.62
	MEM		1		90%	68.81%	81.38
	LC			1	90%	100.00%	92.46
Two Resource Weighted	vCPU & MEM	1/2	1/2		44%	72.06%	61.88
	MEM & LC		1/2	1/2	90%	70.44%	88.34
	vCPU & LC	1/2		1/2	42%	100.00%	68.65
High Resource Weighted	vCPU	3/4	1/8	1/8	34%	66.25%	58.19
		1/2	1/4	1/4	33%	65.00%	54.01
	MEM	1/8	3/4	1/8	35%	66.44%	60.00
		1/4	1/2	1/4	33%	65.19%	54.41
	LC	1/8	1/8	3/4	34%	67.00%	57.61
		1/4	1/4	1/2	34%	65.50%	56.72
Equal Weight	vCPU & MEM & LC	1/3	1/3	1/3	35%	67.12%	60.64

TABLE VIII: Mean steady-state output for vFW in 13 resource-scalarisation scenarios.

		vCPU	MEM	LC	vCPU (2 cores)	MEM (1.6 GB)	OR/LC (%)
Single Resource Weighted	vCPU	1			40%	100.00%	92.25
	MEM		1		90%	68.81%	92.50
	LC			1	90%	100.00%	94.45
Two Resource Weighted	vCPU & MEM	1/2	1/2		42%	70.06%	87.50
	MEM & LC		1/2	1/2	90%	69.69%	94.64
	vCPU & LC	1/2		1/2	40%	100.00%	88.84
High Resource Weighted	vCPU	3/4	1/8	1/8	39%	68.75%	85.20
		1/2	1/4	1/4	40%	68.50%	89.10
	MEM	1/8	3/4	1/8	39%	68.62%	83.44
		1/4	1/2	1/4	39%	68.31%	84.91
	LC	1/8	1/8	3/4	40%	69.81%	80.75
		1/4	1/4	1/2	39%	68.38%	82.45
Equal Weight	vCPU & MEM & LC	1/3	1/3	1/3	39%	69.75%	79.57

4) Highlight conclusions & limitations: Link Capacity is more untactful on OR performance in Snort Passive mode than in Snort Inline mode and vFW because traffic is forwarded directly to the destination without being inspected before forwarding. vFW gives the highest OR to LC ratio at around 83.63% compared to Snort Inline mode (58.50%) and Snort Passive mode (57.36%). Because vFW drops packets incoming to unallowed ports and forwards packets from allowed ports, packet delay does not occur in this VNF. But unlike vFW, Snort Passive duplicates packets with a latency stop before forwarding them, and Snort Inline packets must be inspected before being sent to the output link. This inspection delay

causes *congestion in the output link*. When considering the effect of the weights of each resource's reward function on reducing the corresponding resource while maintaining the OR, we find that the behaviour of each VNF is different.

Finally, we acknowledge the following experimental limitations. First, the performance of vFW and Snort can fluctuate under a constant resource allocation, depending upon the number of configuration rules loaded into the system. Our method assumes that all configurable aspects of VNF behaviour, aside from resource allocation, exhibit relative stability throughout the VNF's lifecycle. Future research should assess the performance of RL in scenarios involving dynamic configuration changes. Second, iPerf has limited traffic generation capabilities, e.g., it struggles to reach rates $\geq$1 Gbps, and packets are not completely realistic. While a well-configured iPerf suffices for demonstrating the proposed method and showcasing an experimental proof of concept, it cannot fully capture a production network deployment with real traffic.

VI. Conclusion

We introduce iOn-Profiler as an intelligent *online learning* VNF profiler using ML and in particular RL, incorporating Q-Learning across a range of optimisation objectives. This autonomous profiling RL model-based adjusts to network dynamics and our work and study results demonstrate its effectiveness by improving the efficiency of profiling for two modes of the Snort (Inline mode and Passive mode) VNF and for vFW, a virtual firewall VNF. We investigate 39 scenarios (13 per VNF type) with different RL resource allocation objective weights to understand the impact of different resource types on the quality of our profiling model's resource allocation decisions. Our comprehensive evaluation results highlight the importance of considering multiple resource optimisation objectives and examining each VNF type individually with online learning, rather than with a statically trained SL model that is impossible to adapt to dynamics such as in demand patterns or be easily used for transfer learning purposes.

Our future research plans include expanding our model for *service function chains* in various VNF configurations and for different VNF types across different network resource substrates. We also aim to further explore network adaptability and the benefits of iOn-Profiler with *transfer learning* between different resources and/or across different VNF types.

Acknowledgement

This work received funding from UK funded Project REASON under the FONRC sponsored DSIT, and EU projects 5G-VICTORI and 5GASP (grant agreements No. 857201 and No. 101016448).

References

[1] R. Nejabati *et al.*, "Zero-touch network orchestration at the edge," in *30th Intern. Conference on Computer Communications and Networks, ICCCN 2021, Athens, Greece, July 19-22, 2021*, pp. 1–5, IEEE, 2021.

[2] S. Moazzeni *et al.*, "A novel autonomous profiling method for the next-generation nfv orchestrators," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 642–655, 2020.

[3] T. Morris, R. Vaughn, and Y. Dandass, "A retrofit network intrusion detection system for modbus rtu and ascii industrial control systems," in *2012 45th Hawaii International Conference on System Sciences*, pp. 2338–2345, IEEE, 2012.

[4] M. C. Luizelli *et al.*, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106, IEEE, 2015.

[5] W. Fang *et al.*, "Joint spectrum and it resource allocation for efficient vnf service chaining in inter-datacenter elastic optical networks," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1539–1542, 2016.

[6] V. Sciancalepore *et al.*, "z-torch: An automated nfv orchestration and monitoring solution," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1292–1306, 2018.

[7] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[8] J. Gou *et al.*, "A generalized mean distance-based k-nearest neighbor classifier," *Expert Systems with Applications*, vol. 115, pp. 356–372, 2019.

[9] V. Verma *et al.*, "Interpolation consistency training for semi-supervised learning," *arXiv preprint arXiv:1903.03825*, 2019.

[10] A. Xu *et al.*, "Applying artificial neural networks (anns) to solve solid waste-related issues: A critical review," *Waste Management*, vol. 124, pp. 385–402, 2021.

[11] S. A. Juliano, "Nonlinear curve fitting: predation and functional response curves," in *Design and analysis of ecological experiments*, pp. 159–182, Chapman and Hall/CRC, 2020.

[12] S. Van Rossem *et al.*, "Profile-based resource allocation for virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1374–1388, 2019.

[13] M. Bunyakitanon *et al.*, "Auto-3P: An autonomous VNF performance prediction & placement framework based on machine learning," *Computer Networks*, vol. 181, p. 107433, 2020.

[14] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-End Performance-based Autonomous VNF Placement with adopted Reinforcement Learning," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2020.

[15] P. Jaisudthi *et al.*, "i-profiler: Towards multi-objective autonomous vnf profiling with reinforcement learning," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops*, pp. 1–6, 2023.

[16] J. O. Iglesias, J. A. Aroca, V. Hilt, and D. Lugones, "Orca: an orchestration automata for configuring vnfs," in *Proceedings of the 18th ACM/IFIP/USENIX middleware conference*, pp. 81–94, 2017.

[17] A. Mestres *et al.*, "A machine learning-based approach for virtual network function modeling," in *2018 IEEE Wireless Comms and Networking Conference Workshops (WCNCW)*, pp. 237–242, IEEE, 2018.

[18] M. Peuster and H. Karl, "Understand your chains and keep your deadlines: Introducing time-constrained profiling for nfv," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 240–246, IEEE, 2018.

[19] S. Van Rossem *et al.*, "VNF performance modelling: From stand-alone to chained topologies," *Computer Networks*, vol. 181, p. 107428, 2020.

[20] M. G. Khan *et al.*, "NFV-Inspector: A systematic approach to profile and analyze virtual network functions," in *2018 IEEE 7th international conference on cloud networking (CloudNet)*, pp. 1–7, IEEE, 2018.

[21] V. R. Chintapalli, V. S. K. Giduturi, B. R. Tamma, and A. A. Franklin, "RAVIN: A Resource-aware VNF Placement Scheme with Performance Guarantees," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2023.

[22] N. Ferdosian *et al.*, "Autonomous Intelligent VNF Profiling for Future Intelligent Network Orchestration," *IEEE Transactions on Machine Learning in Communications and Networking*, pp. 1–1, 2023.

[23] R. Jia *et al.*, "Towards diagnosing accurately the performance bottleneck of software-based network function implementation," in *Passive and Active Measurement*, (Cham), Springer Nature Switzerland, 2023.

[24] M. Peuster and H. Karl, "Profile your chains, not functions: Automated network service profiling in devops environments," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–6, IEEE, 2017.

[25] G. M. Yilma *et al.*, "Benchmarking open source nfv mano systems: Osm and onap," *Computer communications*, vol. 161, pp. 86–98, 2020.

[26] M. Bunyakitanon *et al.*, "HELICON: orchestrating low-latent & load-balanced virtual network functions," in *IEEE International Conference on Communications, ICC 2022, Seoul, Korea, May 16-20, 2022*, pp. 353–358, IEEE, 2022.