Meta Relational Learning-Based Service-Tailored VNF Deployment for B5G Network Slice

Zexi Xu, *Student Member, IEEE*, Lei Zhuang, Weihua Zhuang, *Fellow, IEEE*, Yuxiang Hu, Wenshuai Mo, and Zihao Wang

***Abstract*—To bring 5G systems and networks to life in large-scale commercial applications, academia community has started the research beyond 5G (B5G), in which network slicing (NS) is proposed as a new paradigm for building service-tailored B5G networks. In each network slice, to precisely control the service quality and cost, deploying the service-required virtual network functions (VNFs) by utilizing the linkage between the characteristics of this slicing task and the characteristics of different servers in the B5G network is essential. Therefore, aiming at gaining the ability of learning and adapting new tasks quickly and cost effectively, we view the NFV deployment problem as a meta relational learning process that explores the meta mapping relation between service-tailored slicing tasks and the B5G physical network and propose a service-tailored VNF deployment framework, abbreviated as StailNet. Instead of training a one-strategy-fits-all deployment model, we focus on "learning" how to train a deployment model and propose to learn the features of servers and slicing tasks from the perspective of knowledge graph-based representation learning, then locate the initial meta mapping relation by extracting meta information in the task-agnostic meta space and exploring the service-tailored meta mapping relation in the task space for each task, so that we can quickly obtain the solution by a few gradients on the initial meta mapping relation. To highlight the performances of StailNet, we do comprehensive simulations. Simulation results demonstrate that our StailNet outperforms the selected representative algorithms in the literature.**

***Index Terms*—Beyond 5G, Internet of Things (IoT), knowledge graph representation learning, meta-learning, network slicing (NS), virtual network functions (VNFs) deployment.**

I. Introduction

THE EXPONENTIAL growth of new businesses, such as mobile video services (e.g., YouTube and TikTok), Internet of Things (IoT)-based services, and Industry 4.0, has triggered global initiatives toward developing the fifth-generation (5G) mobile systems. Since 2020, aiming at transforming the one-size-fits-all service manner to the one-size-per-service manner, academia community has started the emphasis on research beyond 5G (B5G), where various edge devices and terminals (such as vehicles and drones) [1] have evolved as key supplementary parts of B5G networks, and they are placing higher requirements on the network Quality of Service (QoS).

To support different types of services, B5G relies on the concept of network slicing (NS) [2]. Each network slice is composed of flows and network functions (NFs [3], e.g., firewalls, monitors, and load balancers) required by the service, which can be performed over the same substrate network as an end-to-end logical "dedicated network," customized for a specific B5G use case. Thanks to NF virtualization (NFV) [4], NFs have been decoupled from the dedicated substrate hardware and implemented as software-defined virtual NFs (VNFs) instances, supporting on-demand service provisioning in each slice. Therefore, as the main enabler of NS, NFV provides an opportunity to determine how to flexibly deploy the service-required VNFs in each B5G network slice, so that the service-specific requirements can be satisfied and QoS can be further enhanced [5].

However, deploying VNFs in a B5G network slice is more complicated than solely deploying VNFs in a core network (e.g., core cloud or datacenters) or in an edge network (e.g., edge cloud, edge cloudlet, or datacenters), as we have to consider deploying VNFs in both core servers (with relatively sufficient resource capacity and high latency) and edge servers (with limited resource capacity and low latency) [6]. So, to improve the quality of B5G service, we need to utilize the linkage between the characteristics of the slicing tasks and the characteristics of the different servers in B5G network. For example, the requirements of autonomous driving network services are ultralow latency and high reliability, while the requirements of 4K/8K HD video network services are high speed and throughput [7], thus autonomous driving network slices need to place more VNFs in edge servers, and 4K/8K HD video slices need to deploy the required VNFs in resource-sufficient core servers. Although based on extracting the features of slicing tasks, there has been some progress [8], [9], [10], [11], [12], [13], [14], [15] using train and test-based intelligent algorithms, such as reinforcement learning [16] and deep reinforcement learning [17] to solve the VNF deployment problem, these methods of obtaining one-strategy-fits-one/one-strategy-fits-all model are costly when switching from one task to another. Because the relationship between physical server

Manuscript received 6 April 2023; revised 9 June 2023, 4 July 2023, and 18 July 2023; accepted 6 August 2023. Date of publication 9 August 2023; date of current version 7 December 2023. This work was supported by the Science and Technology Major Project of Henan Province (Song Shan Laboratory) under Grant 221100210900-03. *(Corresponding authors: Lei Zhuang; Weihua Zhuang.)*

Zexi Xu, Lei Zhuang, and Zihao Wang are with the School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China (e-mail: 1277160576@qq.com; ielzhuang@zzu.edu.cn; iewangzihao@163.com).

Weihua Zhuang is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: wzhuang@uwaterloo.ca).

Yuxiang Hu is with the Institute of Information Technology, PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China (e-mail: chxachxa@126.com).

Wenshuai Mo is with Zhengzhou University, Zhengzhou 450001, China (e-mail: mws7931@gs.zzu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2023.3303452

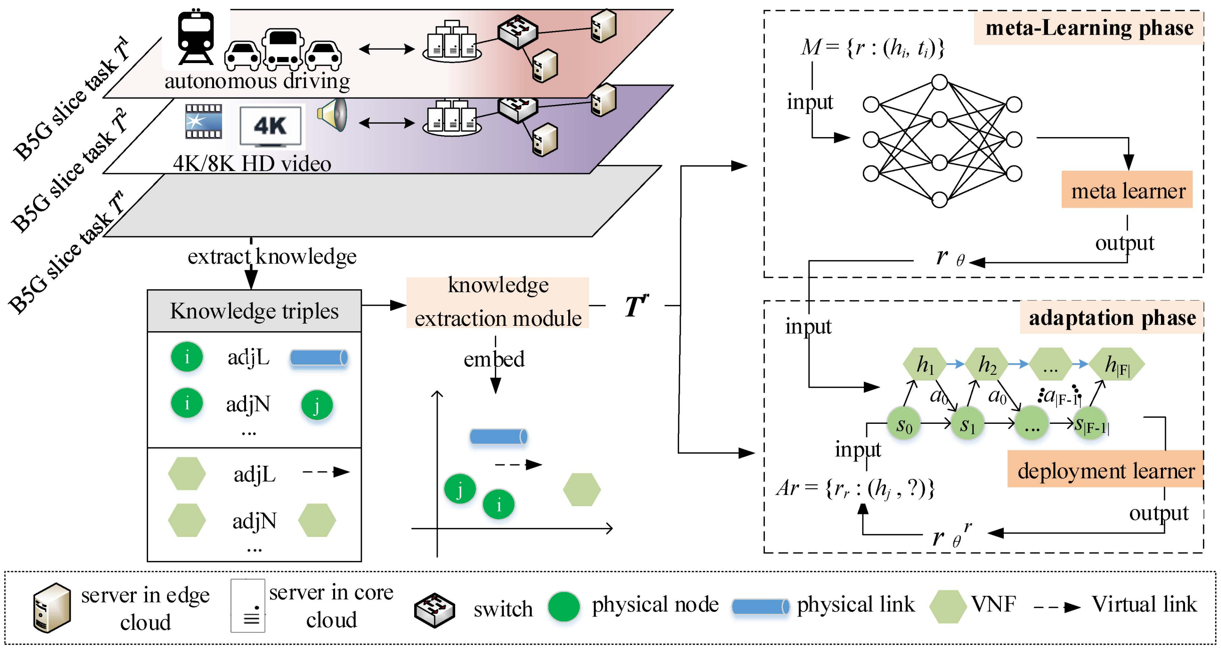


Fig. 1. Overall architecture of StailNet.

features and slicing task features is not considered in VNF deployment, their trained model that performs well on one task and wants to perform well on a slicing task with completely different characteristics also requires a lot of model training on completely new data.

Therefore, in this article, we focus on "learning" how to train a deployment model cost effectively based on a meta-learning process [18], rather than focusing on training a model for specific task as traditional goal-driven algorithms do. And aiming at gaining the ability of learning and adapting new tasks quickly and cost effectively, we propose a meta relational learning framework for service-tailored VNF deployments among different B5G service slices, abbreviated as StailNet. To simplify the learning process, we tackle the deployment problem from a novel perspective based on the intuition that the most critical information to be transferred from the existing deployment instances to the new slicing task should be based on the shared relational knowledge [19], which captures the relationship between physical server characteristics and slicing task characteristics on VNF deployment. We call such relation service-tailored meta mapping relation, marked as r. Moreover, we call the information that facilitates the acquisition of the shared knowledge as "meta gradient." In this article, meta gradient is defined as the loss gradient of meta mapping relation which will be used to make a rapid update when transferring meta mapping relation among different tasks. The overall architecture of StailNet is shown in Fig. 1. Especially, as shown in Fig. 1, to embed the features of slicing tasks and the B5G physical network in the same vector space for meta mapping relation extraction and learning, StailNet is equipped with a knowledge extraction module based on the knowledge graph representation learning method, transH [20]. Then, StailNet solves the service-tailored VNF deployment by dividing into two phases, namely, meta learning and adaptation, which are helpful in the following two perspectives, respectively: 1) learning and transferring meta gradient from observed example tasks in the task-agnostic meta space to locate the initial meta mapping relation and 2) accelerating the adaptation to different tasks by supporting meta-level continual learning in the task space. Eventually, we can quickly converge to the solution by applying a few gradients on the initial meta mapping relation and with using only a few training data and iterations.

The remainder of this article is organized as follows. Section II discusses the related works and the motivation for this study. Section III presents the B5G network slice model and discusses service-tailored VNF deployment in a B5G slicing task. In Section IV, technical details of the *StailNet* are presented. Section V provides the performance evaluation of the proposed solution, and Section VI concludes this study.

II. Related Works and the Contributions for This Study

In this section, we first review the state-of-the-art works dealing with the VNF deployment problem. Then, we elaborate on the contributions for this study by borrowing the concept of meta-learning.

A. VNF Deployment

Recent years have witnessed the proliferation of the studies on the VNF deployment problem for achieving some specific optimization objectives. For example, in [9] and [10], mathematical programming methods, such as integer linear programming (ILP) [12] and mixed ILP (MILP) [13], were used to solve this optimization problem. A preliminary work was proposed in [9], aiming at jointly minimizing the maximum network link utilization and the number of CPU cores used by the instantiated VNFs, Addis et al. formulated an NFV network model suitable for ISP operations and devised an MILP formulation to solve the generic VNF chain routing optimization problem. The heuristics [11], [12], [13], such as the greedy method-based [21], Markov model-based [22], and so on, are proposed to calculate the approximate optimal deployment. For instance, de Freitas Bezerra [11] formulated VNF deployment as a multiobjective problem and designed two optimization methods, the nondominated ranking genetic algorithm and the differential evolutionary algorithms, to calculate the approximate optimal solution.

Moreover, with the booming development of artificial intelligence, recently intelligent methods [14], [15] have become the most popular methods for VNF deployments and have achieved more outstanding performance in this optimization problem than the previous two approaches. Typically, aiming at jointly minimizing the operation cost of NFV providers and maximizing the total throughput of accepted requests, Xiao et al. [23] proposed an online policy gradient-based deep reinforcement learning approach to automatically deploy SFCs, in which an MDP model is introduced to capture the dynamic network state transitions. Wang et al. [24] proposed a deep reinforcement learning-based approach for adaptive NFV-RA by combining both the graph convolution network and sequence-to-sequence model to generate placement strategies.

B. Contributions

In summary, further research is required to integrally address the service-tailored VNF deployment in B5G slicing task as mentioned in Section I. Most existing solutions are customized for a specific business or application scenario. Despite the efficiency of business-customized solutions, their service-specific constraints make them difficult to be flexibly applied to other network scenarios. Although some train and test-based intelligent algorithms [23], [24] can automatically deploy VNFs based on historical experience, they also face a dilemma: as for supervised learning [25], it is not difficult to achieve scenario-by-scenario personalized deployment, if we can provide enough training data for each task, but the fact that the arrival of network operations is usually unpredictable, making this approach too costly for the online deployment problem. As for unsupervised learning [26], it can directly output deployment solutions by feeding newly arriving requests into a model already trained based on historical data, but with the proliferation of new applications, a model that cannot adapt dynamically may perform poorly. In addition, these intelligent approaches need to explore the solution in a huge action space [27], which introduces some unnecessary complexity, because the deployment of different slicing tasks is highly tendentious, e.g., we unnecessarily search for the optimal deployment of the autonomous driving slicing task in the core server.

Therefore, aiming at gaining the ability of learning and adapting new tasks quickly and cost effectively, rather than training a one-strategy-fits-all network model, we view the NFV deployment problem as a meta relational learning process that makes learning simple and intuitive to explore the meta mapping relation between each slicing task and the physical network. Based on the idea of "learn to learn" in meta-learning, we solve this problem by focusing on transferring meta information, enabling the model to learn the most important meta relation and adapt faster. Consequently, our main contributions are as follows.

1) We first propose a time-slot-based B5G network slice model for VNF deployment, which includes both edge cloud servers and core cloud servers with real-time status, and slicing tasks with different requirements that arrive in real time. Then, we present the definition of knowledge base, and with the B5G knowledge base, the definition of service-tailored VNF deployment in a B5G slicing task is given.
2) To achieve our objective—learning and adapting new slicing tasks quickly and cost effectively, we propose a meta relational learning framework, abbreviated as StailNet, for the service-tailored VNF deployment among different B5G business slices. In particular, StailNet is divided into two phases, namely, meta learning and adaptation, focusing on learning how to train the service-tailored meta mapping relation for different slicing tasks. And to embed the information of different dimension spaces into the same vector space for meta information extraction and learning, StailNet is also equipped with a knowledge extraction module based on the knowledge graph representation learning method.
3) Extensive experiments are conducted to demonstrate the convergence and efficiency of knowledge extraction in StailNet. And trace-driven evaluations on a typical B5G network topology verify the proposed VNF deployment framework StailNet superior performance in terms of average acceptance ratio, cost, and revenue by using only a few data points and training iterations.

III. Network Modeling

In this section, we first introduce the time-slot-based B5G network slice model. Then, we present the task formulation of StailNet. Key notations are listed in Table I.

A. B5G Network Slice Model

In B5G networks, it is different from solely deploying requests in core network or edge network, we have to consider placing VNFs in both edge cloud servers and core cloud servers so as to satisfy some strict QoS requirements. Therefore, to implement a B5G physical network, we model the network as a connected undirected graph $G_s = (N_s, L_s)$ along with $|N_s|$ server nodes and $|L_s|$ links. Especially, the set of core nodes is represented as N_{sc}, and the set of edge nodes is represented as N_{se}, as well as there are several levels of switches for ensuring the connectivity of the nodes. To deal with the real-time network variations, we consider that the network slices deploy executes in an online manner, where the set of time slots is $T_m = \{1, 2, \ldots, |T_m| - 1\}$, and each time slot $t \in Tm$ starts when a slicing task is arrived and terminates when this task is accepted or rejected to deployment. At

TABLE I
KEY NOTATIONS

Symbol	Description
slice model	
$G_s = (N_s, L_s)$	The B5G physical network
$\|N_s\| = N_{sc} \cup N_{se}$	The set of physical nodes, where N_{sc} is the set of core nodes, N_{se} is the set of edge nodes
$\|L_s\|$	The set of physical links
T_m	The set of time slots
$C(n_t) = (C_{cpu}(n_t), C_{mem}(n_t))$	The available resource capacity of physical node n at time t in term of CPU and memory
$C_{bw}(l_s^t)$	The available bandwidth on link l_s at time t
$T^r = (F, L_v)$	The group of B5G tasks, where F is the set of VNFs in the rth slice, L_v is the traffic flow between two VNFs
$D(v) = (D_{cpu}(v), D_{mem}(v))$	The resource demand of VNF v in term of CPU and memory
$d(v_i)$	The latency requirement of VNF v_i
$d(n_j)$	The latency on server n_j
$D_{bw}(l_v)$	The bandwidth requirement of link l_v
t_s^i	The arriving time of request T^r
t_d^i	The service time of request T^r
$K = \{E, P, L, TP\}$	The knowledge base, where E, P, L, TP are the sets of entities, properties, literals and triples, respectively
(h, r, t)	The triple of deployment information, where h is a VNF, r is the mapping relation, t is a physical node
θ	Parameters of the service-tailored meta mapping relation r for a task T^r
meta-training phase	
M	Input The meta-training data set
$r_{\hat{\theta}_{meta}}$	Output The initial meta mapping relation
adaptation phase	
A^r	Input The adaptation data set of T^r
$r^r : (h, ?)$	Output The VNF deployment solution, where r^r is the service-tailored meta mapping relation for T^r, h are the service-required VNFs in T^r, ? are the servers computed to host VNFs h

each time slot t, each physical node $n \in N$ in G_s has an available resource capacity (i.e., computing resources and memory resource), marked as $C(n^t) = (C_{\text{cpu}}(n^t), C_{\text{mem}}(n^t))$, note that other types of resource, such as storage, can also be added in $C(n^t)$ if necessary. While, for each link $l_s \in L_s$ in G_s, $C_{bw}(l_s^t)$ denotes the available transmission resources between server nodes n_i and n_j, where $i \neq j$ and $i, j \in N$.

With respect to the virtual slices, we consider a group of B5G tasks T that requires $|T|$ different slices. The business traffic is generated by the different tasks. Each network task T^r ($r \in |T|$) requests a specific set of VNFs, we adopt the graph theory to model them and label the rth slicing task as $T^r = (F, L_v)$, where $F = \{v_1, v_2, \ldots, v_{|F|}\}$ is the set of service-required VNFs in the rth slice, and each $v_i \in F$ has a resource demand in terms of CPU and memory, denoted by $D(v_i) = (D_{\text{cpu}}(v_i), D_{\text{mem}}(v_i))$. And if a VNF $v_i \in F$ is deployed, it requests the queuing delay and processing delay on node $n_j \in N_s$, marked as $d(v_i) > d(n_j)$. While L_v is the set of virtual links for data forwarding, and we assume that a traffic flow $l_{v_{ij}}$ is injected into the network through a server node n_i, and leaves from a server node n_j. The flow on each segment must be forwarded on links that meet certain bandwidth requirements, denoted as $D_{bw}(l_v)$. In virtualization and slicing research, each slicing task arrives, following the known Poisson distribution, and has an arriving time t_s^i and a duration time t_d^i, representing how long request T^r is expected to be in service after its slice be placed.

B. Task Formulation of StailNet

In this section, we first present the formal definition of the knowledge base and denote the deployment information in B5G knowledge base. Then, with a B5G knowledge base, we give the definition of the service-customized VNF deployment for a B5G slicing task T^r and formulate the overall architecture of StailNet.

Definition 1 (Knowledge Base K): The content of a particular domain or field of knowledge. In a knowledge base K, each fact is stated in a triple of the form (*entity*, *property*, *value*), in which value can be either a literal or an entity. The sets of entities, properties, literals, and triples are denoted by E, P, L, and TP, respectively.

For example, the deployment information for a B5G slicing task can be stated in the triples of the form (h, r, t), where h represents one of the VNFs in a slice, which provides the deployment policy for h to t, r represents the mapping relation for this VNF deployment, and t represents a physical node in the physical network that already hosts or tends to host this VNF. On this context, the service-tailored VNF deployment in a B5G slicing task T^r can be defined as follows.

Definition 2 (Service-Tailored VNF Deployment in a B5G Slicing Task T^r): With a B5G knowledge base K^B, predicting the tail entity in the newly arrived slicing task T^r linked with an adaptive mapping relation r^r to head entity, formulated as $r^r : (h, ?)$, is called service-tailored VNF deployment in the slicing task T^r.

As defined above, after transforming the unstructured network information to uniformly structured information - triples in the knowledge base K^B, a service-tailored deployment task T^r can be always defined for a service-tailored mapping relation. So intuitively, we can figure out how to obtain a deployment strategy by learning how the relational information to be learned and transferred from the existing deployment instances to the new slicing task, to learn and adapt new tasks quickly and cost effectively.

Specifically, we train the service-tailored r^r with two data sets. The first one is meta-training set $M = \{(h^M, t^M) \in E^B \times E^B | (h^M, r_{\hat{\theta}_{\text{meta}}}, t^M) \in TP^B\}$, which is the support set consisting of several randomly selected deployment instances in K^B to learn and transfer meta information from the task-agnostic meta space. And $r_{\hat{\theta}_{\text{meta}}}$ is marked the initial meta mapping relation. The second one is adaptation set $A^r = r^r : \{(h^A, t^?) \in E^B \times E^B | (h^A, r_{\theta^r_{\text{meta}}}, t^?) \in TP^B\}$ for slicing task T^r, which is the query set consisting of all triples to be predicted to learn and transfer meta information from the task space. And in A^r, h_i is the feature vector of the ith VNF in T^r, $t^?$ is the solution for hosting this VNF, r^r is the service-tailored meta mapping relation, which can obtain by fine-tuning the initial meta mapping relation according to the features of this task. With these two data set, the meta-optimization is performed over parameters θ of mapping relation r, whereas the solution is computed using service-tailored r.

In effect, by optimizing the parameters of the r_θ-based deployment strategy, the proposed StailNet can take one or a small number of gradient steps on the initial meta mapping relation to converge quickly to the final solution. Thus, the major difference between StailNet and the previous train and test-based deployment is that while both are based on empirical learning, the latter is focused on training a model that performs well on certain objectives (e.g., revenue and request acceptance rate), while the former is focused on learning how to train a model based on a meta relational learning process, so that we can achieve the objective quickly and cost effectively.

Accordingly, the service-tailored VNF deployment in StailNet can be divided into two phases: meta learning and adaptation, learning the initial $r_{\widehat{\theta}_{\text{meta}}}$ and fine-tuning it to obtain the service-tailored r^r, respectively. As shown in Fig. 1, in meta learning, meta learner, the agent of this learning phase, learns and transfers meta information from the meta training set M and acquires initial metamapping relations r by a simple meta-training process

$$r_{\widehat{\theta}_{\text{meta}}} = \underset{\theta}{\operatorname{argmax}} \frac{1}{n} \sum_{k=1}^{n} \mathop{\mathbb{E}}_{\tau \sim T^k} \left[r_\theta^k \right] \tag{1}$$

$$\widehat{\theta}_{\text{meta}} = \underset{\theta}{\operatorname{argmax}} \frac{1}{n} \sum_{k=1}^{n} \mathop{\mathbb{E}}_{\tau \sim T^k} \left[\ell_\theta \left(T^k \right) \right] \tag{2}$$

where θ is the parameters of r, r_θ^k is the mapping relation of task T^k in M, $\widehat{\theta}_{\text{meta}}$ is the parameters of the initial meta mapping relation $r_{\widehat{\theta}_{\text{meta}}}$, τ is the triple in T^k, and n is the number of tasks in M. $\ell_\theta(T^k)$ is the objective evaluated on T^k. Particularly, in order to train meta learner to learn how to get the relation r, the loss gradient of $l_r(\theta)$ is calculated and labeled as $\nabla_{r_\theta^k} \ell_\theta(T^k)$, which acts as the meta gradient in the task-agnostic meta space for guiding the learning of meta learner. Accordingly, the initial meta mapping relation $r_{\widehat{\theta}_{\text{meta}}}$ will be obtained by the multitask objective that is guided by the representation loss of tasks in M.

In the adaptation phase, deployment learner, the agent of this learning phase, learns and transfers meta information from the adaptation set A^r, i.e., the specific task space. As shown in Fig. 1, when adapting to a new task T^r, knowledge extraction module of StailNet extracts the knowledge of the substrate network and the task newly arrived (i.e., T^r) and embeds the knowledge of them in the same vector space. Then, the deployment learner extracts the feature of T^r and provides the meta learner with a task-specific feedback in the form of higher order meta information to explain its own status in the current task space and update the parameter θ_r of the mapping relation r

$$\theta_r \leftarrow \widehat{\theta}_{\text{meta}} - \beta \nabla_\theta \sum_{i=1}^{|F^r|} \mathcal{L}_{r_\theta} \left(A^r \right) \tag{3}$$

where the step size β is fixed as the hyperparameters. $|F^r|$ is the number of incomplete triples in the A^r, i.e., the number of VNFs to be deployed in task T^r. $\mathcal{L}(*)$ is the loss function evaluated on the VNF deployment strategy of T^r, implying that the parameters θ in deployment strategy are trained by optimizing the performance of each VNF deployment in T^r. Moreover, in order to train deployment learner to learn how to get the service-tailored mapping relation r^r, the task loss gradient, i.e., $\nabla_\theta \mathcal{L}_{r_\theta}(A^r)$, is used as the meta gradient to guide the learning of deployment learner. Accordingly, the updated meta mapping relation r_θ will be obtained by the task objective and guided by the representation loss of entities in A^r

$$\min_\theta \mathcal{L}\left(T^r\right) = \min_\theta \sum_{i=1}^{|F^r|} \mathcal{L}_{r_\theta} \left(A^r \right) \tag{4}$$

where $\mathcal{L}(A^r)$ is the sum of query loss during training.

Eventually, we can obtain the deployment solution for T^r with the service-tailored r_r by: $? = h + r^r$.

IV. Proposed Service-Tailored VNF Deployment Framework StailNet

In this section, we introduce each module of our StailNet in detail.

A. Knowledge Extraction Module

Same with previous intelligent deployment algorithms, we first convert the information of NS tasks and physical network to uniformly structured data. However, to make those intelligent algorithms, originally developed for Euclidean structure data, applicable to solve graph-like problems, almost all current studies have used the form of resource matrices (e.g., adjacency matrix and edge table) to characterize the network information, which poses a significant limitation: formulating mapping relation r involves two different dimensional spaces, the slicing task and the physical network, so resource matrix-based vector calculation is always complex and have unavoidable biases. So the matrix-based representation approach is not conducive for us to explore the learning of meta-relations.

To reduce the knowledge extraction bias and establish the relationship between the data of different dimension spaces, instead of using the traditional resource matrix-based network representation method, we represent the network in a semi-structured way with flexible patterns—constructing knowledge graphs by extracting knowledge in the slice to be deployed and the physical network, marked as KG_f and KG_s, respectively. The semi-structured information of them is given in Table II. Notably, we mainly focus on the relations between nodes and links in the networks for converting the network data to a set of knowledge triples (eh, s, et), where eh and et represent two different vertices or links in a knowledge graph, and s represents a kind of structure relation between eh and et.

In addition, in this work, we use $\triangle$ to denote the set of golden triplets. Hence, we use $(eh, s, et) \in \triangle$ to state "(eh, s, et) is correct" and $(eh, s, et) \in \bar{\triangle}$ to state "(eh, s, et) is incorrect." We use lowercase bold-face letters to denote the vector representations of the corresponding terms, e.g., ($\mathbf{eh}$, $\mathbf{s}$, and $\mathbf{et}$) denotes the vector representation of triple (eh, s, et). We use capital bold-face letters to denote matrices, and we use superscripts to denote different knowledge bases. For example, $\mathbf{E}^{(1)}$ denotes the representation matrix for entities in K^1 in which each row is an entity vector $\mathbf{e}^{(1)}$.

Next, after extracting information by knowledge graph, we embed information from these two different dimensional

TABLE II
STRUCTURED INFORMATION OF KNOWLEDGE GRAPH KG_f AND KG_s

knowledge graph	schema	knowledge description
KG_f	(eh_TYPE: v_i, s_TYPE: $adjN$, et_TPYE: v_j)	v_i and v_j are the two VNFs in this slice ($i \neq j$); $adjN$ indicates that there is a directed virtual link from v_i to v_j
	(eh_TYPE: v_i, s_TYPE: $adjL$, et_TPYE: l_{vi})	v_i is a VNF in this slice; and l_{vi} is a virtual link in this slice. $adjL$ indicates that l_{vi} is an adjacent link of v_i.
KG_s	(eh_TYPE: n_i, s_TYPE: $adjN$, et_TPYE: n_j)	n_i and n_j are the two servers in the physical network ($i \neq j$); $adjN$ indicates that there is a undirected physical link from n_i to n_j
	(eh_TYPE: n_i, s_TYPE: $adjL$, et_TPYE: l_s)	n_i is a server in the substrate network; and l_s is a physical link in the substrate network. $adjL$ indicates that l_s is an adjacent link of n_i.

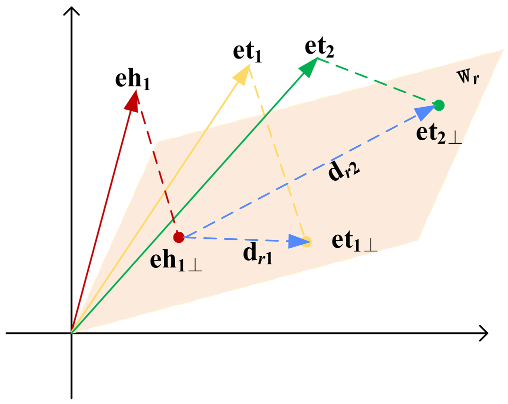


Fig. 2. Knowledge graph embedding.

spaces into the same vector space based on a widely used knowledge graph representation learning method TransH [28]. As illustrated in Fig. 2, for each structure relation s, we first define a hyperplane $\mathbf{w}_r$ (the normal vector). Then, we project the embedding $\mathbf{eh}$ and $\mathbf{et}$ to the hyperplane $\mathbf{w}_r$, where the projections are denoted as $\mathbf{eh}_\perp$ and $\mathbf{et}_\perp$, respectively. Accordingly, the s-specific translation vector $\mathbf{d}_r$ will be positioned in hyperplane $\mathbf{w}_r$ rather than in the same space of entity embedding, so that we can represent reflexive/one-to-many/many-to-one/many-to-many relationships in the s-specific hyperplane. For example, as shown in Fig. 2, there is a head entity $\mathbf{eh}_1$ identified in red is projected on the hyperplane $\mathbf{w}_r$, marked as $\mathbf{eh}_{1\perp}$, if it represents a physical node in the physical network, the tail entities $\mathbf{et}_1$ and $\mathbf{et}_2$ represented in yellow and green represent the two neighboring nodes of this physical node, then the blue dashed lines $\mathbf{d}_{r1}$ and $\mathbf{d}_{r2}$ represent the projection of $adjN$ between them on $\mathbf{w}_r$, of course we can also replace the physical nodes in this example with physical links, virtual links, and VNFs. Thus, for a triple $(eh, s, et) \in \triangle$, $\mathbf{et}\perp$ can be deduced by $\mathbf{eh}\perp$ and $\mathbf{d}_r$ as

$$\mathbf{et}_\perp = \mathbf{eh}_\perp + \mathbf{dr}. \tag{5}$$

Furthermore, we define a scoring function $f(eh, s, et)$ to measure the plausibility of the triple (eh, s, et), which is defined as

$$f(eh, s, et) = \|\mathbf{eh}_\perp + \mathbf{d}_r - \mathbf{et}_\perp\|_2 \tag{6}$$

where $\| * \|_2$ is the L2 norm of a vector. By restricting $\|\mathbf{w}_r\|_2 = 1$, we can easy to get $\mathbf{eh}_\perp = \mathbf{eh} - \mathbf{w}_r^\top \mathbf{eh}\mathbf{w}_r$, $\mathbf{et}_\perp = \mathbf{et} - \mathbf{w}_r^\top \mathbf{et}\mathbf{w}_r$. Eventually, the scoring function $f(eh, s, et)$ is

$$f(eh, s, et) = \left\| \left(\mathbf{eh} - \mathbf{w}_r^\top \mathbf{eh}\mathbf{w}_r\right) + \mathbf{d}_r - \left(\mathbf{et} - \mathbf{w}_r^\top \mathbf{et}\mathbf{w}_r\right) \right\|_2. \tag{7}$$

Algorithm 1: Training of Knowledge Graph Embedding

```
Input: $KG_S$, average number of tail entities per head entity $tph$, the average number of head entities per tail entity $hpt$, margin $\gamma$, vector dimensionality $k$, learning rate $\alpha$, iteration times $epochnum$, batch size $b$
Output: $\mathbf{V}^{es}$= { entity vectors $\mathbf{n}$, $\mathbf{l}$, relation vector $\mathbf{d}_r$, normal vector $\mathbf{w}_r$ }
1  extract $numt$ positive triples;
2  conduct $numt$ negative triples by random visit a positive triple and corrupt it by replacing the head with probability $\frac{tph}{tph+hpt}$, replacing the tail with probability $\frac{hpt}{tph+hpt}$ training;
3  randomly generate $\mathbf{V}^{es}$={$\mathbf{n}$, $\mathbf{l}$, $\mathbf{d}_r$, $\mathbf{w}_r$ };
4  while $epoch < epochnum$ do
5      Load training set $S$=positive triples, negative triples;
6      while $batch \leq \lceil \frac{num_t}{b} \rceil$ do
7          sample $b$ triples from the remaining $S$, and remove them from $S$;
8          update $\mathbf{V}_{es} = \mathbf{V}_{es} - \alpha \times \frac{1}{b}\sum_{(eh,r,et)} \nabla[f(eh,r,et) - f(eh',r,et') + \gamma]_+$;
9          $\mathbf{V}_{es} = \frac{\mathbf{V}_{es}}{\|\mathbf{V}_{es}\|}$;
10     $epoch = epoch + 1$, $batch = batch + 1$
11 return $\mathbf{V}$es;
```

From this, the embedding training process of KG_s is shown in Algorithm 1 and since the relationships we consider are the same, we can implement the KG_f embedding by replacing the data set of KG_s in the input, which will not be repeated here. The core concept of it is based on the following steps.

At the first step, in order to keep the representation bias under control, we not only add the gold knowledge directly extracted from KG_s into the training set but also generate some negative triplets by replacing the head or tail entities in the golden triples with probabilities $(tph/[tph + hpt])$ and $(hpt/[tph + hpt])$, respectively. We label the negative triples as $(eh', s, et) \setminus (eh, s, et')$ and insert them into the training set (lines 1–3 of the algorithm).

For a positive triple, we expect $f(eh, r, et) \approx 0$, while for a negative triplet, we expect $f(eh, r, et) \gg 0$. So, after we load training set S, which includes both the positive triples and negative triples (line 5 in Algorithm 1), we use the following margin-based ranking loss to train this model by discriminating between positive triples and negative triples:

$$L = \sum_{(eh,s,et)\in\triangle} \sum_{(eh',s,et')\in\bar{\triangle}} \left[f(eh, s, et) + \gamma - f\left(eh', s, et'\right)\right]_+ \tag{8}$$

where $[x]_+ = \max(0, x)$, γ is the margin separating positive and negative triplets, which is a hyperparameter.

Then, as shown in lines 4–10, we adopt stochastic gradient descent to minimize L with the following constraints:

$$\frac{|\mathbf{w}_r^\top \mathbf{d}_r|}{\|\mathbf{d}_r\|_2} = 0 \ \forall r \tag{9}$$

$$\|\mathbf{w}_r\|_2 = 1 \ \forall r \tag{10}$$

where (9) guarantees the translation vector $\mathbf{d}_r$ is in the hyperplane.

Finally, as the gradient is computed and the model parameters are updated after a mini-batch. We can obtain the vector representations of the nodes, links, and their relations.

B. Meta Training

To extract the meta gradient from observed example tasks in M on the fly, the meta learning phase is designed to learn the initial mapping relation $r_{\widehat{\theta}_{\text{meta}}}$ based on experience, which can be viewed from a feature learning standpoint as building an internal representation of the deployment instances. Algorithm 2 shows the pseudocode of this process implemented by the meta learner, the agent of this learning process.

Computationally, we first sample a task T^e from M and extract entity-pair specific relation r via an L-layers fully connected neural network

$$\mathbf{x}_0 = \mathbf{h}_{\text{vi}} \oplus \mathbf{t}_{\text{si}}$$

$$\mathbf{x}^l = \sigma\left(\mathbf{W}^l x^{l-1} + b^l\right)$$

$$\mathbf{r}_{(\mathbf{h}_{\text{vi}},\mathbf{t}_{\text{si}})} = \mathbf{W}^l \mathbf{x}^{l-1} + b^l \tag{11}$$

where $\mathbf{h}_{\text{vi}}$ and $\mathbf{t}_{\text{si}}$ are embeddings of head entity $h_{\text{vi}} \in VNF$ and tail entity $t_{\text{si}} \in N$ in T^e with dimension k, respectively, which have been vectorized in knowledge extraction module. L is the number of layers in this neural network and $l \in \{1, \ldots, L-1\}$. $\mathbf{W}^l$ and b^l are weights and biases in layer l. $\mathbf{y}\oplus\mathbf{z}$ represents the concatenation of vectors $\mathbf{y}$ and $\mathbf{z}$. And we use LeakyReLU for activation σ. Notably, for ease of presentation, all parameters related to r will be denoted by θ in the following.

Then, for a one-shot task T^e, meta learner can directly obtain the mapping relation of it, represented by $\mathbf{r}^k = \mathbf{r}_{(\mathbf{h}_{\text{vi}},\mathbf{t}_{\text{si}})}$. While for an N-shot task (i.e., there are more than one VNF to be deployed in this task), meta learner generates the mapping relation $\mathbf{r}^e$ via averaging all $\mathbf{r}_{(\mathbf{h}_{\text{vi}},\mathbf{t}_{\text{si}})}$, which can be calculated as

$$\mathbf{r}^e = \frac{\sum_{i=1}^N r_{(h_{\text{vi}},h_{\text{si}})}}{N}. \tag{12}$$

Algorithm 2: Meta Training

```
  Input: $h_{vi}$; $t_{si}$; $\gamma$; Learning rate $\varphi$
  Output: $r_\theta^r$
1 Sample a $T^e$ form $M$;
2 Compute $r_\theta^e$ by Eq. (11);
3 for all $T^k$ in $M$ do
4     Compute $\ell_\theta^k(T^k)$ by Eq. (13);
5     Evaluate $\nabla_{r_\theta^k} \ell_\theta^k(T^k)$ (i.e., representaion meta
      gradient);
6     update $\theta_k'$ by Eq. (15);
7 Compute $\widehat{\theta}_{meta}$ by Eq. (2);
8 return $\widehat{\theta}_{meta}$;
```

Next, to make $\mathbf{r}^e$ more representative for different tasks, meta learner further optimizes it by a meta-training process, where a score function is defined to evaluate the truth value of entity pairs for the current task by applying the key idea of knowledge graph embedding methods [29]. And we calculate the score for each entity pair $(\mathbf{h}_{\text{vi}}, \mathbf{t}_{\text{si}})$ in $T^k \in M$ as follows:

$$S(\mathbf{h}_{\text{vi}}, \mathbf{t}_{\text{si}}) = \left\|\mathbf{h}_{\text{vi}} + \mathbf{r}_\theta^k - \mathbf{t}_{\text{si}}\right\|. \tag{13}$$

Meanwhile, a representation loss function $\ell_\theta(T^k)$ is defined to acquire the meta gradient in the task-agnostic meta space so as to update the parameters of r_θ through per task in M

$$\ell_\theta\left(T^k\right) = \sum_{(h_{\text{vi}},t_{\text{si}})\in T^k} \left[\gamma_l + S(\mathbf{h}_{\text{vi}}, \mathbf{t}_{\text{si}}) - S\left(\mathbf{h}_{\text{vi}}, \mathbf{t}'_{\text{si}}\right)\right]_+ \tag{14}$$

$$\theta_r \leftarrow \theta_e - \varphi \nabla_{r_\theta^k} \ell_\theta^k\left(T^k\right) \tag{15}$$

where $\ell_\theta(T^k)$ should be small which represents the task T_θ^k is properly represented by the specific r^k, $\nabla_{r_\theta^k} \ell_\theta^k(T^k)$ is the representation loss gradient, which acts as the meta gradient in this meta learning phase to explain how should the mapping relation r be updated.

Eventually, as shown in lines 7–10 of Algorithm 2, by observing the representation loss gradient of each sampled T^k in M, meta information can be summarized into the parameters of the initial mapping relation $r_{\widehat{\theta}_{\text{meta}}}$ as shown in (2) $(\widehat{\theta}_{\text{meta}})$, and the mapping relation vector can have a rapid update and converge as (1) by minimizing $\sum_{T^k\in M} \ell_\theta(T^k)$.

C. Adaptation

Although in the meta-training phase, we have got the abstraction of the initial mapping relation $r_{\widehat{\theta}_{\text{meta}}}$ in the task-agnostic meta space, we are still some distance from "service-tailored deployment," as the real-time interaction information between the newly arrived slicing task and the physical network has not yet been passed to the learner. Therefore, for a newly arrived task T^r with an adaptation set A^r, in order to feedback higher order meta information to meta learner to quickly locate its status and obtain the fine-tuned r_θ, according to the dependencies between head entities (i.e., VNFs) in the adoption set A^r, we formulate the prediction process of tail entities (i.e., physical nodes that host VNFs) in the adaptation set A^r as a discrete-time finite horizon discounted Markov decision process (MDP), which can be denoted by

$$s_{t+1} = f^{r^r}(s_t, a_t) \tag{16}$$

where $s_t \in \mathbb{R}^A$ are the states and $a_t \in \mathbb{R}^{KG_s}$ are the actions. The dynamics f^{T^r} is parameterized by the mapping relation r^r_θ. At each step, the deployment learner takes an action $a \in A$ according to the policy f^{T^r}. Then, the state s_t will transit to the next state $s_{t+1} \in S$ and an immediate reward $RW^r(a_t, s_t)$ will be generated, which can be formulated over a finite time-horizon as

$$RW^r(a, s) = \mathbb{E}\left[\sum_{t=0}^{|A^r|} |RW^r(a_t, s_t)| s = s_0, a = a_0, s_t = r_\theta(a_t)\right]. \tag{17}$$

Accordingly, in our scenario, we define the three components of MDP in the task space as follows.

1) State Definition: In the tth time, we define the state $s_t \in S$ as the current state of server nodes, which can be described as

$$s_t = \left(\mathbf{e}_{t1}, \mathbf{e}_{t2}, \ldots, \mathbf{e}_{t|N|}\right) \tag{18}$$

where s_t indicates the environmental state at time t that needs to be fed back to the meta learner. And $\mathbf{e}_{ti}$ is the state vector of the server node, which consists of the structural information embedded by the knowledge extraction module and the information about the number of resources available to the physical node at the current moment.

2) Action: The deployment learner has to decide how to deploy the slicing task T^r. It includes two steps: ① predicts tail entities in the adaptation set A^r to assign server nodes to VNFs of T^r and ② chooses suitable links to connect. As the knowledge of the physical network has been embedded in the same vector space as the slicing task. The deployment learner executes an action $a \in (NI, LI)$ at each step, where NI and LI are the set of server indexes and the set of physical link indexes, respectively.

3) Reward Function: To obtain the deployment solution adapted to the newly arrived task quickly and cost-efficiently, in the adaptation phase, we perform meta-optimization from the initial mapping relation $r_{\widehat{\theta}_{\text{meta}}}$ to fine-tune the parameters of it and we formulate the reward function as

$$RW = \lambda * \sum_{l_v \in L_v} \frac{D_{bw}(l_v)}{x * C_{bw}\left(r_\theta\left(l_v^t\right)\right)} + \mu * \omega\left(\tau; A^r, M\right)\ell_\theta\left(T^r\right) - \nu * \left(1 - \widehat{ESS}\right)\left\|\theta - \widehat{\theta}_{\text{meta}}\right\|_2 \tag{19}$$

where λ, μ, and ν are the custom constant, namely, the reward coefficient. The first term computes the off-policy updates on the r^r_θ of new task (data in adaptation set A^r) to ensure the performance, where $x * C_{bw}(r_\theta(l_v^t))$ represents the bandwidth resources required to deploy l_v. The second term performs the parameter updates on old data (i.e., data in meta learning set M) to locate the initial meta mapping relation. $\omega(*)$ is the propensity score, a simple logistic classifier used by Fujimoto et al. [30], which represents the similarity between the data in A^r and the data in the meta-training set M. The third term is an automatically adapting proximal term that prevents degradation of the policy during adaptation, where $\widehat{ESS}$ is the effective sample size [31] between A^r and M that is a measure of the similarly of the new task with the meta-training tasks, which can be calculated by a heuristics Monte Carlo method as: $\widehat{ESS} = (1/|A^r|)([(\sum_{i=1}^{|A^r|}\omega(\tau_i))^2]/[\sum_{i=1}^{|A^r|}\omega(\tau_i)^2]) \in [0, 1]$, where $\omega(\tau) = e^{-\omega^{*\top}\tau}$.

To this end, to deploy the newly arrived slicing task T^r cost-efficiently. We set our training objective based on (4)

$$\min_\theta \mathcal{L}_{r_\theta}\left(A^r\right) = \begin{cases} -RW(T^r), & \text{The solution is feasible} \\ +\infty, & \text{Otherwise.} \end{cases} \tag{20}$$

And we fine-tune $r_{\widehat{\theta}_{\text{meta}}}$ for several epochs to make it adaptable to the newly arrived slicing deployment task T^r as shown in Algorithm 3. In each iteration, for each h_{vi}, i.e., the v_i, the real-time state matrix is first extracted from the substrate network via the knowledge extraction module (line 5), then the deployment learner calculates the selected probability for each physical node, in which the probability p_{s_i} is computed as

Algorithm 3: Adaptation

```
Input: $h_{vi}$; $t_{si}$;$\lambda$, $\mu$ and $\nu$; Number of epochs $numEpoch$;
       Learning rate $\alpha$
Output: $r^r_\theta$
1  while $t \le T$ do
2      for each $T^r$ newly arrived do
3          while iteration < numEpoch do
4              for $VNF \in A^r$ do
5                  extracted the real-time state matrix $s_t$ by
                   knowledge extraction module;
6                  calculate the probability distribution $p_{s_i}$
                   by Eq. (21);
7                  Take an action $a_t$ based on $p_{s_i}$;
8                  Execute action $a_t$ and receive a new state
                   $st+1$:$s_t = r_\theta(a_t)$;
9                  Storage the immediate reward $RW(a_t, s_t)$;
10             if $isMapped(\forall VNF \in T^r)$ then
11                 Apply breadth-first $LinkMap(T^r)$;
12                 if $isMapped(\forall VNF \in T^R, \forall l_v \in T^k)$ then
13                     Compute loss $\mathcal{L}_{T^r}(r_\theta)$ and task loss
                       gradient (i.e., task meta gradient) as
                       $\nabla_\theta \mathcal{L}_{T^r}(r_\theta)$;
14                 else
15                     clear the stacked gradients;
16                 ++iteration;
17             apply task loss gradients to $\theta_r$ by Eq. (3);
18         Update $r^r_\theta$ by Eq. (4);
19         compute $t_{si} = h_{vi} + r^r_\theta$;
20 return $r^r_\theta$;
```

$$p_{s_i} = \begin{cases} \frac{S(h_{v_i}, t_{s_i})}{\sum_{s_i} S(h_{v_i}, t_{s_i})}, D(v_i) < C(n_{s_i}) \\ 0, D(v_i) < Cap(n_{s_i}) \end{cases} \tag{21}$$

and we choose the one with the highest probability to take action as well as calculate immediate reward (lines 6–8). We repeat this process until all the VNFs in T^k are assigned and then process link mapping by applying a breadth-first search to find the shortest paths (lines 10–15). At the end state, the deployment learner calculates the total RW and $\mathcal{L}_{r\theta}(A^r)$. Then, we defined a tensor for the gradient of every step using the stochastic gradient descent optimizer to obtain the task loss gradient $\nabla_\theta \mathcal{L}_{r\theta}(A^r)$ to minimize $\mathcal{L}_{r\theta}(A^r)$. After several epochs, we can stack a batch of gradients, then we apply it to update parameters of r_θ automatically as (3). In the end, the algorithm outputs the final solution of the slicing task or returns information about failing in deployment.

V. Evaluation

In this section, to demonstrate the effectiveness of our proposed VNF deployment method, we simulate the experiments on an online NS deployment environment and deeply analyze the experiment results. The experiment is conducted on a CEC physical network topology with 161 nodes (including 101 servers and 60 switches) generated by the GT-ITM[1] tool, where $N_c = 53$ and $N_e = 48$. We have used the global-wide backbone, transit network Tinet as the core network topology, where all nodes of the backbone network have equal, but finite amounts of resource capacities to host VNFs. And we have introduced three cloud data centers (DCs) (connected to New York, Prague, and Frankfurt) with a fat-tree ($k = 4$) structure in the Tinet topology to simulate the edge network. Same as [12], we configure the core cloud servers in core network with 30–40 units of CPU resource and 30–40 units of Memory resource, while edge cloud servers in DCs with 10–20 units and 10–20 units. The bandwidth resource between any server and ToR switch is set to 60, the delay of them is 3, the bandwidth of each physical link between ToR and aggregation switch is set to 80, the delay 2, and between aggregation and core switch is set to 1000, the delay is 1. The bandwidth resource between core cloud servers is set to 100–150, and the delay is 1–10. The constant value for nodes and links energy consumption is set to $P_l = 150$ (W), $P_b = 150$ (W), and $P_n = 15$ (W).

A. Knowledge Extraction Results

We first extract the key information of the physical network topology according to Table II and transform it into 1959 entities and 7740 golden triplets. And then, in the knowledge extraction module, we set dimension $k = 100$, margin $\gamma = 1$, batch size $b = 5$, and $epoch = 100$ to train our embedding model for transforming the unstructured network information to embedding vectors. Fig. 3 shows the convergence of embedding loss function values under different learning rates α and epochs. As the epoch gets bigger, all loss functions can be seen to sharply drop first and subsequently fluctuate around

[1] GT-ITM. https://www.cc.gatech.edu/projects/gtitm/.

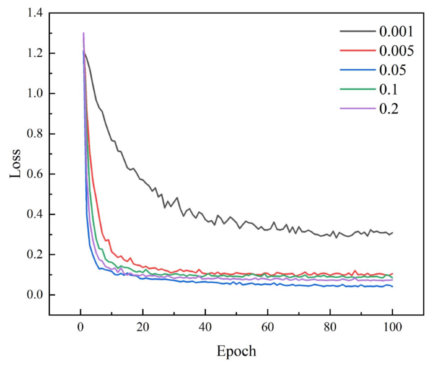


Fig. 3. Loss functions with different α.

TABLE III
Part of Knowledge Extraction Results

dataset	entities
physical network	$n_{sc0} = 0.81449944, 0.3745496, \ldots, 0.85478187$
	$l_{s0} = 1.1132241, -1.1062038, \ldots, 2.339856$
	$\ldots$
slicing task	$v_0 = -0.33547553, 0.5956694, \ldots, 0.902138$
	$v_1 = 0.06818318, 0.2696302, \ldots, -0.46224838$
	$v_2 = 0.19532254, 0.80371743, \ldots, 0.6969126$
	$v_3 = -0.016816698, 0.4335314, \ldots, -0.965447$
	$l_{v1} = 0.06818318, 0.2696302, \ldots, 0.902138$
	$l_{v2} = 0.7272514, -0.20422861, \ldots, -0.46224838$
	$l_{v3} = 0.19532254, 0.80371743, \ldots, 0.6969126$
	$l_{v4} = -0.016816698, 0.4335314, \ldots, -0.965447$

various constants. Larger α speeds up convergence, however if is more than a certain value, such in this example $\alpha = 0.1$ and $\alpha = 0.2$, the loss curves converge to larger values and the corresponding representations are underfitting. Hence, we set $\alpha = 0.05$ and with such parameters setting, our knowledge extraction module can output the vectors of physical nodes and physical links. We show some of them in Table III.

We also embed the key information of the slice to be deployed into the same vector space dynamically. Furthermore, the slicing tasks are generated in the same way as stated in [5]. Each slice consists of different numbers of VNFs from 3 to 8 according to the uniform distribution. Table III shows the embedding results for a simple slicing task example with four VNFs and four virtual links that connect them in turn. As with the physical network, we first transformed it into 8 entities and 12 golden triples, then we call Algorithm 1 to generate negative samples based on these golden triples and constructed the training set consisting of both positive and negative samples, finally, the knowledge extraction module outputs the vectors of these VNFs and virtual links.

And the embedding results of relations $adjN$ and $adjL$ are

$$adjN = [-0.11773149, 0.1236721, \ldots, -0.064747885]$$
$$adjL = [0.49432778, -1.6670148, \ldots, 1.825491]$$

TABLE IV
COMPARING STAILNET TO GRC, DRL, AND METAP

Strategy	StaiNet	GRC	DRL	MetaP
feature extraction	Knowledge graph	–	graph convolution network	knowledge graph
training	meta learning	–	deep reinforce learning	meta learning
deploying	fine-tuning the origion $r_{\widehat{\theta}_{meta}}$ through the adaptation phase	non-recursive greedy resource	by a trained constant model	using the origion $r_{\widehat{\theta}_{meta}}$ directly

B. VNF Deployment Results

Experimental Setup: Our simulation experiments are executed on a computer with 2.40-GHz Intel Core i5 9300h CPU and 16-GB RAM. In the meta-training phase, M includes four example tasks that are randomly selected from the real-world traces from Alibaba [32]. In the adaptation phase, learning rate $\alpha = 0.05$, $numEpoch = 10$, and $batch\_size = 10$. The whole model architecture is built with Tensorflow and Adam optimizer is employed to update the parameters of neural networks. To evaluate the performance of StailNet in online VNF deployment, there are 2000 slicing tasks arrive at the system sequentially according to a Poisson process with an average arriving rate of 40 per 1000 time units. Each slice has a lifetime exponentially distributed with an average of 40. For intuitively showing the experiment results, we average the results in all figures (i.e., from Figs. 4 to 10). The experiments set the data collection point at the beginning of the time window, and one time window is equal to 5000 time units, the experimental simulation is executed within ten time windows. In addition, the virtual links between VNFs obey a random distribution of $[10, 20]$ and $[20, 50]$ for the maximum data rate and delay requirements, respectively. The main parameters of slicing task are the same as discussed in [5], and in order to verify the adaptability of the algorithm to handle a variety of demanding tasks, we scaled up the parameters of VNFs by a factor of two.

Baseline Algorithms: We first use a classical algorithm, the nonrecursive greedy resource allocation (GRA) algorithm [33], as the baseline, which deploys VNFs at the nodes with the most sufficient resources. To evaluate the performance of modules in StailNet, we compare it with DRL [24] and MetaP, where DRL is a typical deep reinforcement learning-based deployment algorithm with the goal of maximizing the long-term average revenue and MetaP is a comparison experiment designed in this article to verify the effectiveness of learners in StailNet. Table IV provides a high-level comparison of StailNet and those methods, listing their strategies.

Performance Evaluation: Fig. 4 reveals the results of the average acceptance ratio for the ten time windows. As plotted, StailNet is able to accept almost all slicing tasks that arrive at the system, achieving a significant advantage. When the system reaches a steady state, the acceptance ratio of GRC and DRL is around 60% and 85%. This benefits from the deployment learner of StailNet, which can accurately extract the features of a new task using knowledge extraction module, and then fine-tune the empirically obtained $r_{\widehat{\theta}_{\text{meta}}}$ to get the service-tailored mapping relation r_θ. So we can derive the efficient deployment solution based on r_θ. Our comparative experiment

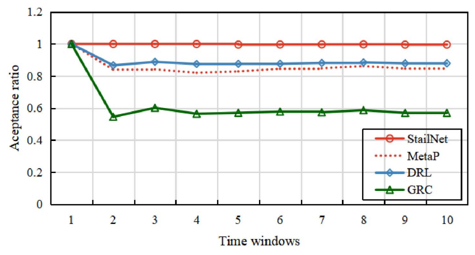


Fig. 4. Average acceptance ratio.

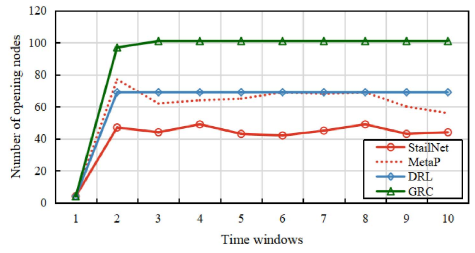


Fig. 5. Average number of opening nodes.

MetaP also demonstrates the effectiveness of the deployment learner designed for the adaptation phase. As shown in Fig. 4, metaP achieves an approximate performance in the average acceptance ratio with DRL, which is also based on a constant deployment model, due to the lack of the adaptation phase. In addition, this result is also attributed to the fact that the service-tailored deployment model of StailNet is more adapted to the complex topological environment of B5G networks. GRC and DRL do not take into account the tendency of different task deployments when facing a B5G network with core and edge servers, so some slicing tasks may be deployed on inappropriate servers, and as network resources are continuously opened and occupied the average acceptance ratio may be decreased.

We also give a cost analysis to highlight the superiority of StailNet. Fig. 5 reveals the variation of the average node openings over ten time windows. The average number of opening nodes directly affects the final average cost and energy consumption, while StailNet always maintains the least number of openings (≈ 50), indicating that our proposed algorithm achieves good performance in terms of resource utilization. As shown in Fig. 6, when the system reaches a steady state, the average number of opening links of the StailNet is less than 100, while the number of GRC and DRL is about 230 and 140, respectively, which are much higher than StailNet. This indirectly reflects that StailNet has the ability to learn what are the actions that most affect the deployment performance with a small amount of data, so it is able to reduce cost

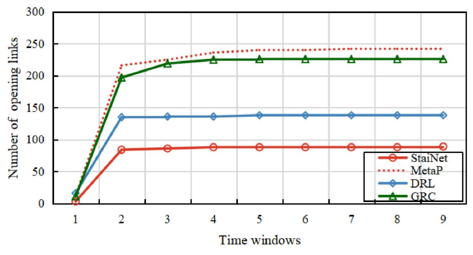


Fig. 6. Average number of opening links.

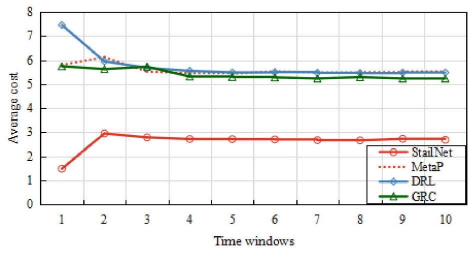


Fig. 7. Average cost.

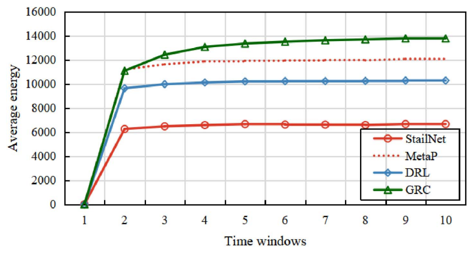


Fig. 8. Average energy.

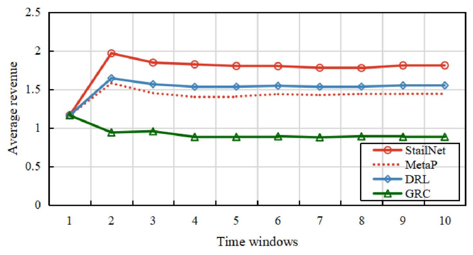


Fig. 9. Average revenue.

and energy by choosing fewer hops to connect nodes. As a result, StailNet always minimizes cost and energy consumption throughout the deployment, as shown in Figs. 7 and 8, respectively. And as shown in Fig. 7, StailNet achieves about 2.1× and 1.8× average revenue reward than the baseline algorithms, which coincides with the results of acceptance ratio and cost, validating the benefits of service-tailored VNF deployment.

Fig. 10 depicts the results of average execution time. As plotted, the execution time for StailNet to deploy a slicing task is slightly higher than that of GRC and DRL. However, there are some facts that cannot be ignored. First, it can be seen from Figs. 4–9 that GRC, as a nonintelligent algorithm performs much worse than other intelligent algorithms. And it is obviously unfair to compare the running time of the GRC without the training process with other intelligent algorithms.

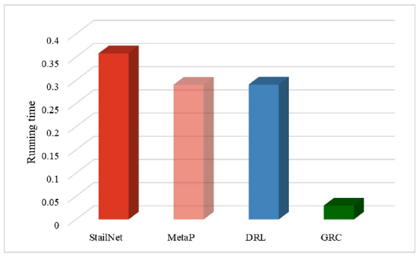


Fig. 10. Average execution time.

As for DRL, DRL is a typical deep reinforcement learning-based deployment algorithm with the goal of maximizing the long-term average revenue. Because the execution time of an online task generally counts the time from the system receiving a newly arrived slicing task to the completion of deployment. Therefore, the execution time of DRL only counts the time spent in the online deployment of a single slice during the testing phase, and the time spent in its offline training is not reflected in the plot. As we all know, the execution time of intelligent algorithms is positively correlated with the size of the training set, the number of iterations, and the batch size of the training process. According to the data provided by DRL, its training set contains 1000 slices with $iteration = 10$ and $batch\_size = 100$, and the training time is 81.62. However, our method contains only four example tasks in the training set with $batch\_size = 10$, thus, although the training and testing are integrated, it still achieves a similar average execution time as DRL, and performs much better. This also demonstrates the efficiency of our approach in terms of learning capability.

VI. Conclusion

In this article, we propose a meta relational learning framework, abbreviated as StailNet. With the goal of gaining the ability of learning and adapting new tasks quickly and cost effectively, we focus on how to "learn" to train a good strategy for different tasks, rather than focusing on how to train a good strategy for individual task as traditional goal-driven algorithms do. Simulation results demonstrate that our StailNet outperforms the selected representative algorithms in the literature.

In future work, we will extract more features for slicing tasks and physical B5G networks and verify the effectiveness of each feature. Meanwhile, we will make efforts to improve the learning efficiency of learners of StailNet and further reduce the cost.

References

[1] F. Irram, M. Ali, M. Naeem, and S. Mumtaz, "Physical layer security for beyond 5G/6G networks: Emerging technologies and future directions," *J. Netw. Comput. Appl.*, vol. 206, Oct. 2022, Art. no. 103431.
[2] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Comput. Netw.*, vol. 167, Feb. 2020, Art. no. 106984.
[3] X. Wu et al., "State of the art and research challenges in the security technologies of network function virtualization," *IEEE Internet Comput.*, vol. 24, no. 1, pp. 25–35, Jan./Feb. 2020.
[4] J. Sun, Y. Zhang, F. Liu, H. Wang, X. Xu, and Y. Li, "A survey on the placement of virtual network functions," *J. Netw. Comput. Appl.*, vol. 202, Jun. 2022, Art. no. 103361.

[5] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware VNF deployment and migration for 5G network slice," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2115–2128, Oct. 2021.

[6] B. Liang, M. A. Gregory, and S. Li, "Multi-access edge computing fundamentals, services, enablers and challenges: A complete survey," *J. Netw. Comput. Appl.*, vol. 199, Mar. 2022, Art. no. 103308.

[7] "Innovative development of 5G application 2022—Whitepaper." 2022. [Online]. Available: http://www.caict.ac.cn/

[8] H. Cao et al., "Toward tailored resource allocation of slices in 6G networks with softwarization and virtualization," *IEEE Internet Things J.*, vol. 9, no. 9, pp. 6623–6637, May 2022.

[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. CloudNet*, 2015, pp. 171–177.

[10] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[11] D. de Freitas Bezerra, "Optimizing NFV placement for distributing micro-data centers in cellular networks," *J. Supercomput.*, vol. 77, no. 8, pp. 8995–9019, 2021.

[12] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient VNF placement for service chaining: Joint sampling and matching approach," *IEEE Trans. Service Comput.*, vol. 13, no. 1, pp. 172–185, Jan./Feb. 2020.

[13] Q. Zhang, X. Qiu, and X. Zhu, "A novel resource optimization algorithm for dynamic networks combined with NFV and SDN," in *Wireless and Satellite Systems*, vol. 281, M. Jia, Q. Guo, and W. Meng, Eds. Cham, Switzerland: Springer, 2019, pp. 283–296.

[14] Z. Zhang, H. Qu, J. Zhao, and W. Wang, "Deep reinforcement learning method for energy efficient resource allocation in next generation wireless networks," in *Proc. CNIOT*, 2020, pp. 18–24.

[15] T. Mai, H. Yao, N. Zhang, W. He, D. Guo, and M. Guizani, "Transfer reinforcement learning aided distributed network slicing optimization in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4308–4316, Jun. 2022.

[16] B. M. H. Zade, N. Mansouri, and M. M. Javidi, "A two-stage scheduler based on new caledonian crow learning algorithm and reinforcement learning strategy for cloud environment," *J. Netw. Comput. Appl.*, vol. 202, Jun. 2022, Art. no. 103385.

[17] G. Zhou, R. Wen, W. Tian, and R. Buyya, "Deep reinforcement learning-based algorithms selectors for the resource scheduling in hierarchical cloud computing," *J. Netw. Comput. Appl.*, vol. 208, Dec. 2022, Art. no. 103520.

[18] M. Chen, W. Zhang, W. Zhang, Q. Chen, and H. Chen, "Meta relational learning for few-shot link prediction in knowledge graphs," in *Proc. Conf. Empir. Methods Nat. Lang. Process. 9th Int. Joint Conf. Natural Lang. Process.*, Hong Kong, Nov. 2019, pp. 4216–4225.

[19] Z. Xu, L. Zhuang, K. Zhang, and M. Gui, "Online placement algorithm of service function chain based on knowledge graph," *J. Commun.*, vol. 43, no. 8, pp. 41–51, 2022.

[20] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017.

[21] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 295–314, Feb. 2021.

[22] H. Cao et al., "Resource-ability assisted service function chain embedding and scheduling for 6G networks with virtualization," *IEEE Trans. Veh. Technol.*, vol. 70, no. 4, pp. 3846–3859, Apr. 2021.

[23] Y. Xiao et al., "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. IWQoS*, 2019, p. 21.

[24] T. Wang et al., "DRL-SFCP: Adaptive service function chains placement with deep reinforcement learning," in *Proc. ICC*, 2021, pp. 1–6.

[25] T. Zoppi and A. Ceccarelli, "Prepare for trouble and make it double! Supervised—Unsupervised stacking for anomaly-based intrusion detection," *J. Netw. Comput. Appl.*, vol. 189, Sep. 2021, Art. no. 103106.

[26] P. Sherubha et al., "An efficient unsupervised learning approach for detecting anomaly in cloud," *Comput. Syst. Sci. Eng.*, vol. 45, no. 1, pp. 149–166, 2023.

[27] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, and R. Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," *J. Netw. Comput. Appl.*, vol. 204, Aug. 2022, Art. no. 103405.

[28] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. AAAI*, 2014, pp. 1112–1119. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531

[29] N. U. A. Bordes and E. A. A. Garc, "Translating embeddings for modeling multi-relational data," in *Proc. NIPS*, 2013, pp. 2787–2795.

[30] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," 2018, *arXiv:1812.02900*.

[31] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, "Meta-Q-learning," 2019, *arXiv:1910.00125*.

[32] "Clusterdata2018." Accessed: Apr. 21, 2020. [Online]. Available: https://github.com/alibaba/clusterdata

[33] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. INFOCOM*, 2016, pp. 1–9.

Zexi Xu (Student Member, IEEE) was born in Zhumadian, Henan, China, in 1997. She is currently pursuing the Ph.D. degree with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China.

She is also a jointly trained Ph.D. student with the University of Waterloo, Waterloo, ON, Canada. Her main research interests include next-generation Internet, network virtualization, and machine learning.

Lei Zhuang was born in Rizhao, Shandong, China, in 1963. She received the Ph.D. degree in computer software and theory from PLA Information Engineering University, Zhengzhou, China, in 2004.

She is currently a Professor and a Ph.D. Supervisor with Zhengzhou University, Zhengzhou. Her main research interests include future network architecture, network virtualization, and model checking.

Weihua Zhuang (Fellow, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from Dalian Maritime University, Dalian, China, and the Ph.D. degree in electrical engineering from the University of New Brunswick, Fredericton, NB, Canada.

Since 1993, she has been a Faculty Member with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, where she is a University Professor and a Tier I Canada Research Chair of Wireless Communication Networks. Her current research focuses on network architecture, algorithms and protocols, and service provisioning in future communication systems.

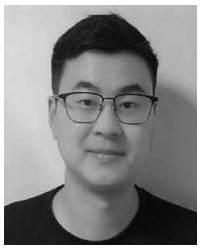
Dr. Zhuang is the recipient of the Women's Distinguished Career Award in 2021 from IEEE Vehicular Technology Society, the R.A. Fessenden Award in 2021 from IEEE Canada, the Award of Merit in 2021 from the Federation of Chinese Canadian Professionals (Ontario), and the Technical Recognition Award in Ad Hoc and Sensor Networks in 2017 from IEEE Communications Society. She is the President and an Elected Member of the Board of Governors of the IEEE Vehicular Technology Society. She was the Editor-in-Chief of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY from 2007 to 2013, an editor of IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS from 2005 to 2009, the General Co-Chair of IEEE/CIC International Conference on Communications in China 2021, the Technical Program Committee (TPC) Chair/Co-Chair of IEEE Vehicular Technology Conference 2017 Fall and 2016 Fall, the TPC Symposia Chair of the IEEE Globecom 2011, and an IEEE Communications Society Distinguished Lecturer from 2008 to 2011. She is a Fellow of the Royal Society of Canada, the Canadian Academy of Engineering, and the Engineering Institute of Canada.

Yuxiang Hu received the Ph.D. degree from PLA Strategic Support Force Information Engineering University, Zhengzhou, China, in 2011.

He is a Professor with PLA Strategic Support Force Information Engineering University. His research interests mainly include future network, routing protocols, switching design, and network security.

Wenshuai Mo was born in Anyang, Henan, China, in 1997. He is currently pursuing the master's degree with Zhengzhou University, Zhengzhou, China.

His main research directions include network virtualization and NFV deployment.



Zihao Wang was born in 1998. He is currently pursuing the master's degree with the School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou, China.

His research interests include next-generation Internet, network function virtualization, and cyberspace security.