

Received July 14, 2021, accepted July 19, 2021, date of publication July 26, 2021, date of current version August 3, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3100283

Gemma: Reinforcement Learning-Based Graph Embedding and Mapping for Virtual Network Applications

MINJAE PARK¹, YOUNGSEOK LEE², IKJUN YEOM¹, AND HONGUK WOO¹, (Member, IEEE)

¹Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

²Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Honguk Woo (hwoo@skku.edu)

This work was supported in part by the Institute for Information and Communications Technology Planning and Evaluation (IITP) (Adaptive Federated Learning in Dynamic Heterogeneous Environment) under Grant 2021-0-00900, in part by the ICT Creative Consilience Program supervised by the IITP under Grant IITP-2020-0-01821, and in part by Samsung Electronics.

ABSTRACT Virtual network mapping (VNM) is a challenge in the field of network virtualization. As VNM variants have been formalized depending on substrate network structures, virtual network specifications, mapping optimization objectives, and other factors, a number of VNM heuristic methods have been introduced. On the other hand, reinforcement learning (RL) algorithms have been incorporated into deep learning frameworks and recognized as a promising solution for solving complex resource allocation problems. In this paper, we present an RL-based graph embedding and mapping framework, **Gemma**, for tackling various VNM problems in a unified end-to-end manner. In the framework, we employ an encoder-decoder deep learning architecture and propose several optimization schemes such as two-stage mapping and model-based selective embedding. Aiming to deal with large-scale VNM problems in both online and offline scheduling systems, the proposed schemes explore the trade-off between inference accuracy and mapping function runtimes, enhancing scalability and timeliness. **Gemma** shows robust performance under various problem conditions, outperforming other heuristic and learning-based methods.

INDEX TERMS Graph Embedding, graph convolution network, reinforcement learning, virtual network mapping, virtual network embedding.

I. INTRODUCTION

In network virtualization, virtual network mapping (VNM) or virtual network embedding is a fundamental function by which virtual networks with resource requests on computing nodes and communication links are mapped onto a substrate network.¹ VNM enables a network operator or data center manager to provision several virtual networks on a single substrate network infrastructure, aiming to utilize underlying resources more efficiently and to maximize the benefits achieved from the infrastructure.

The associate editor coordinating the review of this manuscript and approving it for publication was Ayaz Ahmad¹.

¹In the network virtualization literature, terms such as virtual network embedding, assignment, and mapping (VNE, VNA, VNM) are used in the same meaning. Although we recognize that VNE is commonly used, we use VNM throughout this paper, since we want to confine the term *embedding* to machine learning contexts, i.e., graph embedding.

Much research is focused on specific problem settings on certain types of underlying substrate infrastructure (e.g., data center networks [1], [2], Internet service providers [3], cloud computing [4], and Internet-of-Things [5], [6]) combined with a mapping objective under specific virtual network requirements. Each work presents VNM heuristic algorithms optimized for the context of its specific problem. For example, in [7], the global resource capacity based on the integrated resource features of network nodes and links is used to heuristically determine the ranks of virtual and substrate nodes for allocation and thereby minimize the operation costs of infrastructure providers. In general, heuristic VNM algorithms have high efficiency and performance in solving the original problem. However, they often have limited applicability to differently configured problems, and thus require further engineering to be useful in broader applications.

On the other hand, graph embedding combined with deep neural networks has begun to be accepted as an effective

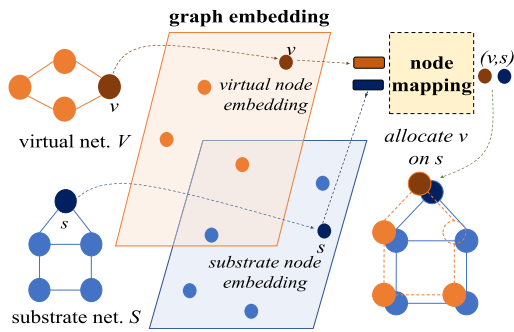


FIGURE 1. Graph embedding and mapping for VNM.

approach to address combinatorial optimization problems. In the context of distributed systems where multiple computing nodes are interconnected and can thus be represented as graphs, several research works have adopted neural graph embedding for applications such as cluster management [8], and telecommunication network management [9], [10]. In principle, neural graph embedding leverages deep learning to convert graph-structured data into low dimensional vectors, with the intention of making it easier to extract useful information from a complex graph representation.

Recently, several research works have introduced learning-based VNM solutions. In [11], DeepVine leveraged convolutional neural networks to extract important embedding features from virtual networks represented in images. In [12], VNM problems were formalized as an Markov decision process (MDP) where continual decisions about selecting target substrate nodes for given virtual nodes are made, and then those problems were addressed through reinforcement learning (RL) such as the Q-learning algorithm. However, these prior works did not fully explore the advantages and complexity of using neural graph embedding to deal with complex virtual network structures or establish fundamental design principles for large scale VNM problems.

In this paper, we present a generic VNM framework, **Gemma** (neural Graph EMbedding and MAPping) that can deal with various VNM problems by exploiting neural graph embedding. We employ RL algorithms to establish an efficient graph embedding and mapping process. Specifically, we incorporate a graph-to-graph recurrent mapping procedure into an encoder-decoder model in which the encoder performs graph embedding tasks for input networks and the decoder performs node-level mapping tasks upon the embeddings. Figure 1 illustrates an example of graph embedding and mapping tasks between virtual network V and substrate network S represented as graphs. Each node in V and S is represented (encoded) into the embedding vector space where the mapping between nodes can be determined (decoded) according to a specific objective; e.g., virtual node v in V is *allocated* to substrate node s in S .

Using the encoder-decoder model, we focus on the applicability of RL techniques to various VNM problems including large-scale online and offline network mapping scenarios.

To address the issue of scale, we develop two optimization schemes to reduce the complexity of the recurrent mapping procedure in **Gemma**. We use a two-stage mapping scheme by which combinatorial scheduling decisions between two networks are made sequential. The scheme makes use of different levels of embeddings to efficiently calculate mapping scores (ranks). Specifically, a rank function for virtual nodes is evaluated to yield a potential top-ranked node v , and then another rank function is evaluated to calculate the mapping score of v on each substrate node. The highest-scored node pair, say (v, s) , indicates that we should allocate virtual node v to substrate node s . Using that procedure, the former rank function exploits not only node-level embedding for each virtual node but also network-level embedding for a substrate network. This simplifies the mapping procedure, while preserving its accuracy.

We also employ a model-based selective embedding scheme that can reduce the computation loads of graph embedding tasks for a time-varying network. This scheme exploits a prediction model about graph embedding changes, i.e., the difference between a pair of embedding data corresponding to before and after the mapping of a virtual node. The scheme is intended to replace iterative and computationally intensive operations for embedding a substrate network with low-cost model inferences, whenever optimistically appropriate. This prediction model is learned individually offline, and is used online when it is integrated with the encoder-decoder model.

The main contributions of our work are as follows.

- We present the **Gemma** framework by which VNM problems can be commonly formulated as a recurrent mapping procedure within a graph embedding space and efficiently addressed using deep RL methods.
- We propose optimization schemes for **Gemma** to expedite the recurrent mapping procedure for large virtual and substrate networks.
- We demonstrate the wide applicability of **Gemma** via simulation tests and show that it can outperform other heuristic and learning-based algorithms in terms of specific VNM objectives under various problem conditions.

The rest of the paper is organised as follows. Section II describes the problem of VNM and our RL-based framework approach to it. Section III presents the structure of our framework which has network encoder and mapping decoder modules, and Section IV describes two optimization schemes for the framework. Section V, Section VI, and Section VII provide the experiment results, the related research works, and the conclusion, respectively.

II. FRAMEWORK OVERVIEW

In the section, we describe the problem formulation of VNM and our RL-based framework **Gemma** for the problem. Figure 2 depicts the framework structure where multiple virtual networks and a substrate network are presented iteratively as input, and an encoder-decoder neural network model generates the node mapping for that input as corresponding

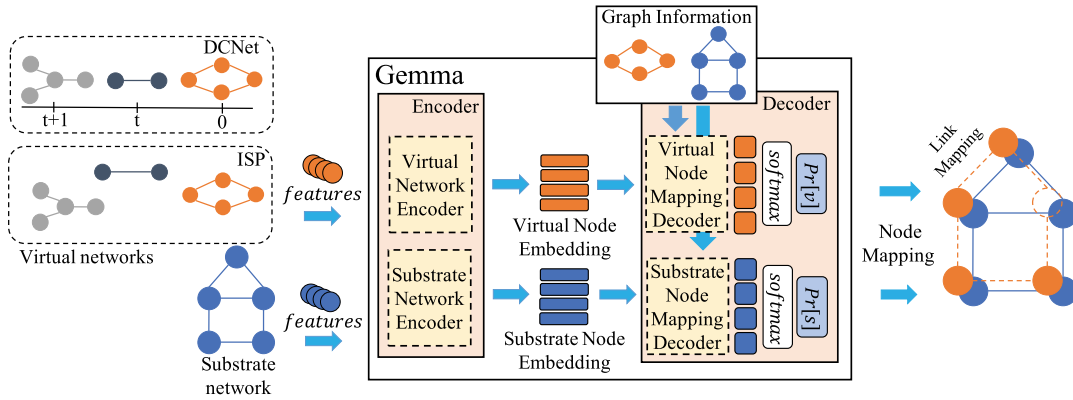


FIGURE 2. The structure of Gemma framework: the framework with a neural encoder-decoder model takes the network features as input and generates a set of mapping node pairs between the input networks to solve VNM problems (i.e., DCNet, ISP). The upper left of the figure depicts that DCNet is an online decision problem of allocating virtual networks that arrive at runtime, whereas ISP is an offline decision problem of allocating a given set of virtual networks at once.

outputs in the context of a specific VNM problem. The node mapping output can then be used as input for a link mapping procedure. **Gemma** incorporates link features in the node mapping procedure through graph embedding, providing an end-to-end learning structure with a specific link mapping algorithm (e.g., shortest path algorithm).

A. VIRTUAL NETWORK MAPPING

In general, VNM can be seen as a resource allocation problem involving complex network structures, as elements (i.e., nodes and links) of virtual and substrate networks are associated with resource requirements and capabilities. VNM algorithms thus intend to establish some optimal conditions. For instance, to ensure high resource utilization and monetary profits, many virtual networks need to be co-hosted on a single substrate network without compromising performance, as a substrate network resource can be partitioned to meet the resource requests of several virtual networks. This optimality varies depending on the objectives of each VNM problem, e.g., acceptance ratio [13], QoS [14], reconfiguration cost [15], or energy-efficiency [3], [16]. Nonetheless, the solutions to all the VNM problems share a similar mapping structure.

The mapping of virtual nodes to substrate nodes manages an n -to-1 node relationship and the mapping of virtual links to substrate links manages an n -to- m link relationship [13]. That is, the physical resources of a substrate node and a substrate link can be partitioned for multiple virtual nodes and links. Furthermore, the physical resources of several substrate links can be combined to generate a substrate path that a virtual link can span.

In this paper, to demonstrate the generic applicability of our RL-based approach, we consider two specific VNM problems with different network characteristics and mapping objectives.

- **VNM problems in a data center network (DCNet)** involve data-parallel tasks that are not known in advance

and arrive at runtime [8]. Each task has multiple sub-tasks with dependency (e.g., Spark data processing jobs) and is represented in a directed acyclic graph (DAG). The time to complete each data-parallel task varies depending on the amount of input data and the underlying resources used. To minimize the completion time of data-parallel tasks, we represent them in virtual networks that can run on a networked system of computing nodes in a data center and formulate resource allocation for virtual networks as a VNM process. In addition, a DCNet provides the resources of a substrate network structured as a certain DCNet topology, e.g., fat-tree [17].

- **VNM problems in Internet service provider (ISP)** involve a bunch of packet processing tasks in a geographically distributed infrastructure. Each task request of network packet processing can be represented in a DAG. This is a common VNM configuration (e.g., network function virtualization) in which the VNM performance improves as the number of virtual networks accepted (allocated) by the underlying substrate network at the ISP increases [18]. To maximize the acceptance rate, our model learns to efficiently allocate resources between networks. Unlike in the DCNet problem, we use the *Erdoes-Renyi* random graph model [19] for ISP substrate networks.

Throughout this paper, we call these representative VNM problems DCNet and ISP. Note that DCNet can be also referred to as an *online* problem and ISP as an *offline* problem with regard to the taxonomy of conventional task scheduling and resource allocation approaches [18].

To address these VNM problems, we consider a networked system structure consisting of a set of virtual networks and a substrate network. Each virtual network V is annotated with the set of resources required by its service, and substrate network S is annotated with its resource capacity. Furthermore, we presume that substrate resources are

consumed in virtual networks, and that they become available again after use. Several resource types such as CPU, GPU, memory, storage on nodes, and bandwidth on links can be specified.

B. NEURAL GRAPH EMBEDDING AND MAPPING

Our framework **Gemma** is intended to establish an optimal policy for allocating multiple virtual networks V on substrate network S according to a VNM objective, while satisfying the resource demands of all the allocated virtual networks within the resource capacity of S . This policy also decides whether each V is accepted or rejected for execution, similar to admission control mechanisms, because substrate resources are necessarily finite.

RL is used to train the encoder-decoder model in **Gemma**, where rewards are acquired according to a given metric specific to a VNM problem. That metric corresponds to the VNM performance obtained by the end-to-end learning of the encoder-decoder model. The encoder transforms the raw features of virtual and substrate networks to their respective vector representations using graph embedding methods, and the decoder iteratively makes inferences from the vector representations to generate a resource allocation plan using rank functions. That is, the encoder represents network nodes in the embedding space, thereby rendering them comparable to each other and consistent in the context of a given VNM problem. These embeddings allow the decoder to decide the mapping in which each virtual node is allocated to a node in a shared substrate network to optimize the VNM performance. We will explain this encoder-decoder model in Section III.

To handle large networks, we employ a two-stage mapping scheme that can reduce the computational complexity of node mapping. In this two-stage strategy, the ranks of virtual nodes are calculated to select the highest-scored virtual node first, and then the ranks of pairs between the selected virtual node and each substrate node are calculated. We use the decoder structure with both node-level and network-level graph embeddings, thus enabling the transformation of combinatorial mapping decisions to two-sequential decisions without significant performance degradation. The decoder structure enables rank estimates close to those that use combinatorial computation, while retaining computational efficiency. We will explain this two-stage mapping in Section IV-A.

Furthermore, to reduce the computation loads of performing graph embedding tasks for a time-varying network, we use an optimistic model-based scheme. Our scheme leverages a supervised learning method to predict the distance between two embeddings for a substrate network obtained before and after mapping a virtual node. Rather than updating the embeddings upon every mapping, which can be computationally intensive, we use the prior embedding unless the expected update is significant. We will explain this model-based selective embedding in Section IV-B.

Algorithm 1 Gemma Framework

```

1: // Input: virtual net.  $V$ , substrate net.  $S$ 
2: unallocated nodes  $\mathcal{O}_k^{(V)} \leftarrow \mathcal{O}^{(V)}$ , mapping plan  $MAP \leftarrow []$ 
3: backup raw matrix  $X_{bk}^{(S)} \leftarrow X^{(S)}$ 
4: virt. embedding  $E^{(V)} \leftarrow \text{NetEmbed}(X^{(V)})$  using Eq. (2)
5: subs. embedding  $E^{(S)} \leftarrow \text{NetEmbed}(X^{(S)})$  using Eq. (2)
6: for  $i \leftarrow 1$  to  $|\mathcal{O}_k^{(V)}|$  do
7:   Update resource  $X^{(S)}$  using Eq. (7)
8:   subs. embedding  $E^{(S)} \leftarrow \text{NetEmbed}(X^{(S)})$  using Eq. (2)
9:    $(v, s) \leftarrow \text{NodeMap}(E^{(V)}, E^{(S)}, \mathcal{O}_k^{(V)})$  using Alg. 2 or 3
10:  if  $(v, s)$  is NULL then
11:     $X^{(S)} \leftarrow X_{bk}^{(S)}$ 
12:    return NULL // unsuccessful mapping
13:  end if
14:   $\mathcal{O}_k^{(V)} \leftarrow \mathcal{O}_k^{(V)} - [v]$ ,  $MAP \leftarrow MAP + [(v, s)]$ 
15: end for
16: return MAP // return the mapping plan for  $V$  on  $S$ 

```

III. EMBEDDING AND MAPPING STRUCTURE

In this section, we illustrate the architecture of **Gemma**. We denote a multi-layer perceptron (MLP) with several affine transform layers as a function,

$$\begin{aligned} \text{MLP}_u(x) &= \text{Affine}_u(\sigma(\text{Affine}_u(\cdots))), \\ \text{Affine}_u(x) &= W_u^{(\theta)}x + b_u^{(\theta)} \end{aligned} \quad (1)$$

where σ is an activation function and u is the name by which the function implementation should be distinguished. Throughout this paper, we generally omit u , assuming that model parameters (e.g., $W^{(\theta)}$ and $b^{(\theta)}$) are properly defined for each learnable function.

Algorithm 1 represents the **Gemma** implementation where **NetEmbed**($\cdot$) and **NodeMap**($\cdot$) correspond to the encoder and decoder respectively.

A. NETWORK ENCODER

In network encoding, a set of raw representations of virtual and substrate networks is transformed into vector representations for further processing to extract relational features. For each virtual network V , the features are specified using a set of resource requirements for nodes and links and are contained in respective raw representations. For a substrate network S , we have the same representation except its features are specified according to its available resource capacities.

In our notation, we use the raw vector representation x for each node. For a virtual node v in V , we use superscript v for its raw vector representation, i.e., $x^{(v)}$. Similarly, for a substrate node s in S , we use $x^{(s)}$. For a network, we use the matrix representation X ; accordingly, we use $X^{(V)}$ and $X^{(S)}$ for V and S , respectively. We also use $\mathcal{O}^{(V)}$ to denote a set of virtual nodes in V , and $\mathcal{O}^{(S)}$ to denote a set of substrate nodes in S .

Consider a network with n nodes. We obtain the node-level graph embedding $E : [e_1, e_2, \dots, e_n]$ for its

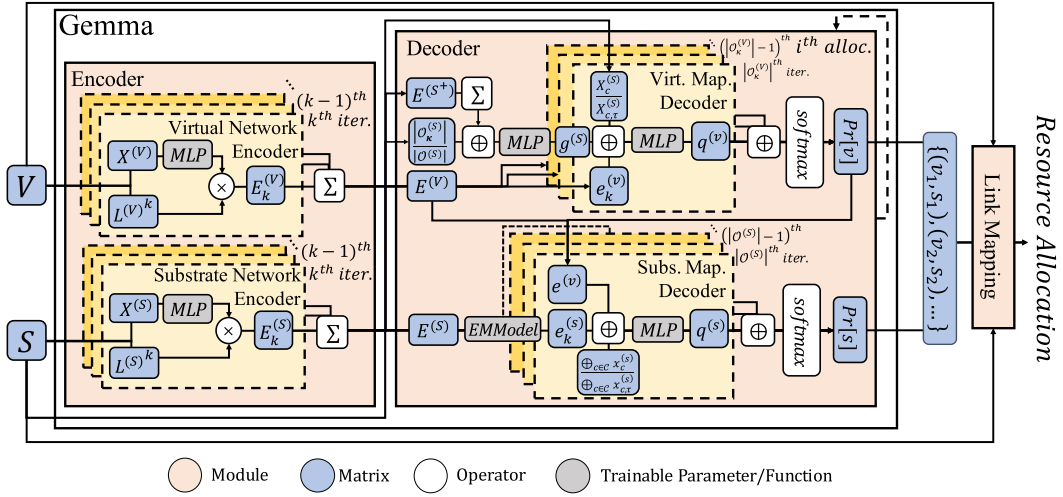


FIGURE 3. The structure of Gemma framework: the framework yields a set of mapping node pairs (virtual node v , substrate node s) for a virtual network V and substrate network S through the encoder-decoder model. The encoder performs embedding for a given network and the decoder performs node mapping, where training both is driven by RL. As shown, the node mapping output is used for link mapping to complete a given VNM task.

raw matrix representation X as,

$$E = \sum_{i=1}^T \text{MLP}(X) \times L^i \quad (2)$$

where L is the Laplacian matrix and T indicates the distance at which the links from a node are considered important. We exploit L to represent the connectivity of nodes in X in the form of an adjacent matrix, and we empirically set the node distance of embedding to $T = 3$. Similar to [20], [21], we observe that the embedding model in which $T = 3$ yields the best performance in our experiments.

In Algorithm 1, each input network is converted into its respective matrix representation by **NetEmbed** using Eq. (2). Embeddings for virtual networks $E^{(V)}$ are time-invariant, while those for substrate networks $E^{(S)}$ are time-varying. Substrate resource capacities change over time depending on the allocations for virtual networks. Thus, the embeddings for S need to be recalculated upon changes of substrate resource capacity to maintain high accuracy. The left part in Figure 3 illustrates the encoder implementation for **NetEmbed**, where two respective sets of MLP model parameters for virtual and substrate networks are learned through the same end-to-end RL training.

B. NODE MAPPING DECODER

The decoder is used to make sequential inferences on the mapping between virtual and substrate nodes. Consider an n -sized embedding $E^{(V)}$ and an m -sized embedding $E^{(S)}$

$$E^{(V)} : [e_1^{(v)}, \dots, e_n^{(v)}], E^{(S)} : [e_1^{(s)}, \dots, e_m^{(s)}] \quad (3)$$

obtained by Eq. (2) for a virtual network V and a substrate network S , where $n = |\mathcal{O}^{(V)}|$ and $m = |\mathcal{O}^{(S)}|$ denote the number of nodes in the virtual and substrate networks respectively. To make inferences from those embeddings,

Algorithm 2 NodeMap in Gemma Plain

```

1: // Input: Virtual and Substrate embeddings  $E^{(V)}$  and  $E^{(S)}$ ,
   // unallocated. nodes  $\mathcal{O}_k^{(V)}$ 
2: // Node mapping for each pair of virtual and substrate nodes
3: calculated ranks  $Q \leftarrow []$ 
4: for each  $v' \in \mathcal{O}_k^{(V)}$  do
5:   for each  $s' \in \mathcal{O}^{(S)}$  do
6:     if Allocable( $v', s'$ ) using Eq. (5) then
7:        $q^{(v', s')} \leftarrow \text{MLP}(e^{(v')}, e^{(s')})$  using Eq. (4)
8:        $Q \leftarrow Q + [q^{(v', s')}]$ 
9:     end if
10:   end for
11: end for
12: if  $Q$  is [] then
13:   return NULL // unsuccessful mapping
14: end if
15: Calculate  $\text{Pr}[(v, s)]$  using Eq. (6) with ranks  $Q$ 
16: Sample  $(v', s')$  pair from  $\text{Pr}[(v, s)]$ 
17: return  $(v', s')$  // return node mapping

```

the decoder calculates the ranks (mapping scores) of virtual and substrate node pairs, which are the basis for establishing the probability for node mapping. Specifically, the rank of each node pair ($v \in \mathcal{O}^{(V)}$, $s \in \mathcal{O}^{(S)}$) is calculated by

$$q^{(v, s)} = \text{MLP}([e^{(v)} : e^{(s)}]) \quad (4)$$

In this calculation, we consider only the set of allocable node pairs, which can be found using an allocable test, i.e.,

$$\text{Allocable}(v, s) = \bigwedge_{c \in \mathcal{C}} x_c^{(s)} \geq x_c^{(v)} \quad (5)$$

where $x_c^{(s)}$ and $x_c^{(v)}$ denote the available capacity of s and the requirement of v , respectively, with respect to a finite resource of type c , and $\mathcal{C}$ denotes a set of resource types. We also maintain a set of unallocated virtual nodes in successive resource allocations, as $\mathcal{O}_\kappa^{(V)}$ in Algorithm 1.

Using the node pair ranks calculated in Eq. (4), we obtain the mapping probability such that v is allocated on s by Softmax,

$$\Pr[(v, s)] = \frac{\exp(q^{(v,s)})}{\sum_{v' \in \mathcal{O}_\kappa^{(V)}, s' \in \mathcal{S}} \exp(q^{(v',s')})}. \quad (6)$$

The rank and probability calculations for virtual and substrate node pairs are conducted iteratively to completely allocate all n virtual nodes in V to S . Each allocation decision must update the resource capacity of S , recalculating $X^{(S)}$ (accordingly, $E^{(S)}$), as the available resources change. For an allocated pair (v, s) and each resource type $c \in \mathcal{C}$, the raw representation $x_c^{(s)}$ for substrate node s is updated by

$$x_{c,i}^{(s)} = x_{c,i-1}^{(s)} - x_c^{(v)} \quad (7)$$

where i denotes the iteration step of node mapping (i.e., line 6 in Algorithm 1). We omit i in our algorithms for simplicity. We represent the node mapping procedure explained above in Algorithm 2, which corresponds to **NodeMap** used in Algorithm 1.

IV. OPTIMIZATION SCHEMES

In this section, we introduce two optimization schemes to reduce the computational complexity of the aforementioned encoding and decoding procedures while maintaining inference accuracy.

A. TWO-STAGE MAPPING

As explained, the rank function for node pairs in Eq. (6) is used to map the virtual nodes to the substrate nodes. In Algorithm 2, when an n -sized virtual network and an m -sized substrate network are given as input, a single node mapping takes $O(nm)$. Here, we present a *two-stage mapping* scheme that exploits two individual rank functions, by which a single allocation of a virtual node on a substrate node can be conducted in linear time with respect to the number of allocable nodes.

In the following, consider embeddings $E^{(V)}$ and $E^{(S)}$ for V and S as in Eq. (3). First, the rank function for virtual nodes calculates a score for each virtual node v . To do so, it exploits not only the node-level graph embedding (i.e., $e^{(v)}$) but also the network-level graph embedding for S . The rank is calculated as

$$q^{(v)} = \text{MLP} \left(\left[e^{(v)} : g^{(S)} : \frac{X_C^{(S)}}{X_{C,\tau}^{(S)}} \right] \right) \text{ for } v \in \mathcal{O}_\kappa^{(V)} \quad (8)$$

where $g^{(S)}$ denotes the network-level graph embedding for S (in Eq. (9)). $X_C^{(S)}$ denotes the resource capacities of S

Algorithm 3 Two-Stage NodeMap in Gemma

```

1: // Input: Virtual and Substrate embeddings  $E^{(V)}$  and  $E^{(S)}$ ,
   unallocated nodes  $\mathcal{O}_\kappa^{(V)}$ 
2: // (a) virtual node mapping
3: Virtual node ranks  $Q^{(V)} \leftarrow []$ 
4: Update capable substrate nodes  $\mathcal{O}_\kappa^{(S)}$  using Eq. (10)
5: if  $\mathcal{O}_\kappa^{(S)}$  is  $\emptyset$  then
   return NULL // unsuccessful mapping
6: end if
7: Update net. embedding  $g^{(S)}$  using Eq. (9) with MLP
8: Update resource capacity  $X_C^{(S)}$  using Eq. (11)
9: for each  $v' \in \mathcal{O}_\kappa^{(V)}$  do
10:   Calculate virt. rank  $q^{(v')}$  using Eq. (8) with MLP
11:    $Q^{(V)} \leftarrow Q^{(V)} + [q^{(v')}]$ 
12: end for
13: Calculate prob. dist.  $\Pr[v]$  using  $Q^{(V)}$  and Eq. (12)
14: Sample virtual node  $v'$  from  $\Pr[v]$ 
15: // (b) substrate node mapping
16: Substrate node ranks  $Q^{(S)} = []$ 
17: for each  $s' \in \mathcal{O}_\kappa^{(S)}$  do
18:   Calculate subs. rank  $q^{(s')}$  using Eq. (13) with MLP
19:    $Q^{(S)} \leftarrow Q^{(S)} + [q^{(s')}]$ 
20: end for
21: Calculate prob. dist.  $\Pr[s]$  using  $Q^{(S)}$ ,  $e^{(v')}$  and Eq. (14)
22: Sample substrate node  $s'$  from  $\Pr[s]$ 
23: Calculate action prob.  $p \leftarrow \Pr[v = v'] \times \Pr[s = s']$ 
24: return  $(v', s'), p$  // return node mapping

```

(in Eq. (11)), and $X_{C,\tau}^{(S)}$ denotes the available resource capacity of S when embedded. Note that $X_{C,\tau}^{(S)}$ is called the embedding-time capacity.

The network-level graph embedding for S continues to be updated for each allocation due to resource changes. Specifically, it is calculated as

$$g^{(S)} = \text{MLP} \left(\left[\sum_{s \in \mathcal{O}_\kappa^{(S)}} e^{(s)} : \frac{|\mathcal{O}_\kappa^{(S)}|}{|\mathcal{O}^{(S)}|} \right] \right) \quad (9)$$

where $e^{(s)}$ is the node embedding for s , $\mathcal{O}^{(S)}$ denotes a set of substrate nodes, and $\mathcal{O}_\kappa^{(S)}$ denotes a subset of $\mathcal{O}^{(S)}$ in which nodes can accommodate at least one unallocated virtual node,

$$\mathcal{O}_\kappa^{(S)} : \{s | s \in \mathcal{O}^{(S)}, v \in \mathcal{O}_\kappa^{(V)}, \exists v \text{ Allocable}(v, s)\}. \quad (10)$$

As S continues to accommodate virtual nodes, its network-level resource capacities must be recalculated, e.g.,

$$X_C^{(S)} = \oplus_{c \in \mathcal{C}} \left(\sum_{s \in \mathcal{O}_\kappa^{(S)}} x_c^{(s)} \right) \quad (11)$$

where $\oplus$ denotes a concatenation yielding a vector. As such, given the multiple resource types in $\mathcal{C}$ that specify the features of nodes and links, we use the composite representation $X_C^{(S)}$. Our case includes resource types such as computing, memory,

and bandwidth, while the properly expressed raw vectors enable the addition of other types. The ranks calculated by Eq. (8) for unallocated virtual nodes are used to determine which virtual node is allocated next. The decision is made by

$$\Pr[v] = \frac{\exp(q^{(v)})}{\sum_{v' \in \mathcal{O}_k^{(v)}} \exp(q^{(v')})} \quad (12)$$

where $\Pr[v]$ represents the probability that v will be selected.

Second, suppose we have a virtual node v for the next allocation as described above. We then calculate the ranks of substrate nodes available to accommodate v by

$$q^{(s)} = \text{MLP} \left(\left[e^{(v)} : e^{(s)} : \frac{\oplus_{c \in \mathcal{C}} x_c^{(s)}}{\oplus_{c \in \mathcal{C}} x_{c,\tau}^{(s)}} \right] \right) \quad (13)$$

for each $s \in \mathcal{O}_k^{(s)}$, where $x_c^{(s)}$ denotes the resource capacities of s with respect to resource type c , (as in Eq. (11)); Similarly, $x_c^{(v)}$ denotes the resource requirements of v , and $\oplus$ denotes a concatenation operator. Then, similar to Eq. (12), we obtain the probability that s is selected to accommodate v from the set of capable substrate nodes $\mathcal{O}^{(s)}$.

$$\Pr[s] = \frac{\exp(q^{(s)})}{\sum_{s' \in \mathcal{O}^{(s)}} \exp(q^{(s')})} \quad (14)$$

In the right part of Figure 3, we represent two-stage mapping: the virtual node mapping decoder corresponding to Eq. (8) and the substrate node mapping decoder corresponding to Eq. (13). We also integrate the procedures for two-stage mapping in Algorithm 3.

Algorithm 4 Model-Based Selective NetEmbed

```

1: if  $\text{EMModel}(E_\tau^{(S)}, X_{C,\tau}^{(S)}, X_C^{(S)}) \geq \theta$  then
2:    $E^{(S)} \leftarrow \text{NetEmbed}(X^{(S)})$  using Eq. (2)
3:    $E_\tau^{(S)} \leftarrow E^{(S)}, X_{C,\tau}^{(S)} \leftarrow X_C^{(S)}$ 
4: else
5:    $E^{(S)} \leftarrow E_\tau^{(S)}$ 
6: end if
```

B. MODEL-BASED SELECTIVE EMBEDDING

When a pair of virtual and substrate networks is given as input, their embeddings are generated through the encoder. While embedding for each virtual network is calculated only once, embedding for a substrate network is calculated at each run due to the time-varying resource features of the substrate network. To mitigate the computation overhead of such embeddings, we leverage a model-based approach. A prediction model estimates the distance between prior and newly updated embeddings for a time-varying network to determine whether to update the embedding data.

Consider embedding $E_{\tau_1}^{(S)}$ and $E_{\tau_2}^{(S)}$ for a substrate network S , which are generated with time-varying resource features at different times, e.g., τ_1 and τ_2 . We define the prediction model, **EMModel**, which yields distance estimates

dst between $E_{\tau_1}^{(S)}$ and $E_{\tau_2}^{(S)}$ (e.g., $|E_{\tau_1}^{(S)} - E_{\tau_2}^{(S)}|$), where only $E_{\tau_1}^{(S)}$ and the respective raw representations $X_{\tau_1}^{(S)}$ and $X_{\tau_2}^{(S)}$ are given as input. The prediction is made without directly calculating $E_{\tau_2}^{(S)}$, which is computationally intensive. Instead, the distance is calculated by

$$\begin{aligned} \text{dst} &= \text{EMModel}(E_{\tau_1}^{(S)}, X_{\tau_1}^{(S)}, X_{\tau_2}^{(S)}) \\ &= \text{MLP} \left(\left[\sum_{s \in \mathcal{O}_{\tau_1}^{(S)}} e_{\tau_1}^{(s)} : \frac{X_{C,\tau_2}^{(S)}}{X_{C,\tau_1}^{(S)}} : \frac{|\mathcal{O}_{\tau_2}^{(S)}|}{|\mathcal{O}_{\tau_1}^{(S)}|} \right] \right). \end{aligned} \quad (15)$$

Algorithm 4 specifies this model-based scheme, where the distance estimate is checked against a given threshold θ . To train **EMModel**, we first collect embedding samples (i.e., $E_{\tau_1}^{(S)}, X_{\tau_1}^{(S)}, X_{\tau_2}^{(S)}$, and $|E_{\tau_1}^{(S)} - E_{\tau_2}^{(S)}|$) using Algorithm 1, and then train **EMModel** with the samples through supervised regression. The code at line 8 in Algorithm 1 is replaced with Algorithm 4 when using model-based selective embedding.

Our proposed framework is based on the overall structure in Algorithm 1 integrated with the new optimization schemes such as the two-stage node mapping in Algorithm 3 and the model-based selective embedding in Algorithm 4. For simplicity, assume $n \approx m$ for an n -sized virtual network and an m -sized substrate network. The complexity of node mapping (**NodeMap**) can be reduced from $O(n^2)$ in Algorithm 2 to $O(n)$ in Algorithm 3. In Algorithm 1, this establishes the mapping complexity of $O(n^3)$ for a single virtual network, which is governed by **NetEmbed** at lines 4 and 5. This is because the model-based selective embedding frequently exploits prior substrate embeddings for **NetEmbed** at line 8 without recalculation.

C. TRAINING GEMMA

Here we show how to train **Gemma** using an RL method in Algorithm 5 and 6. Similar to [8], we employ the modified REINFORCE in Algorithm 5 to maximize the expected rewards by gradient ascent. We also implement a learning environment that simulates resource allocation on a substrate network infrastructure, continuously performing RL actions and calculating rewards over timesteps. Specifically, our environment implementation supports both online and batch scheduling processes to accommodate various VNM problems. For DCNet problems, the environment is set to simulate the processing of virtual network requests that arrive online by continuously allocating and reclaiming substrate resources. In this case, the processing time of a virtual network from allocation to completion is approximated based on the provisioning link ratio, as described in Eq. (19). On the contrary, for ISP problems, the environment considers a bunch of virtual network requests as an offline batch job, thereby allocating several virtual networks in a batch without reclaiming substrate resources.

In general, the raw resource features of input networks can be seen as RL states in **Gemma**. The encoder model takes the features of a pair of virtual and substrate networks (V, S) as input RL states, converting them into the respective

Algorithm 5 Rollout Training **Gemma**

```

1: // Input: model parameters  $\theta$ , dataset  $\mathcal{D}$ 
2: for each episode do
3:   Sample batch  $\mathcal{B} = \{\mathcal{V}, \mathcal{S}\}$  from dataset  $\mathcal{D}$ , init.  $\Delta\theta \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $N$  do
5:     Get reward samples  $R^i$ , prob. samples  $\Pi^i$  using Alg. 6
6:   end for
7:   Get longest episode length  $\mathcal{L} = \max_{i=1}^N (|\Pi^i|)$ 
8:   for  $i \leftarrow 1$  to  $N$  do
9:     Get padding size  $\mathcal{P} = \mathcal{L} - |\Pi^i|$ 
10:    for  $j \leftarrow 1$  to  $\mathcal{P}$  do
11:      // Insert step padding at reward and prob. samples
12:       $R^i \leftarrow R^i + [0]$ ,  $\Pi^i \leftarrow \Pi^i + [1]$ 
13:    end for
14:  end for
15:  for  $k \leftarrow 1$  to  $\mathcal{L}$  do
16:    Compute baseline  $b_k = \frac{1}{N} \sum_{i=1}^N R_k^i$ 
17:    for  $i \leftarrow 1$  to  $N$  do
18:      // Compute gradient and model parameter change
19:       $\Delta\theta \leftarrow \Delta\theta + \nabla_{\theta} \log \Pi_k^i (R_k^i - b_k)$ 
20:    end for
21:  end for
22:   $\theta = \theta + \alpha \Delta\theta$  // model parameter update
23: end for

```

embeddings for further decoding procedures. In Algorithm 6, the **NetEmbed** calls handle the composite of RL states including $X^{(V)}$ and $X^{(S)}$.

On the other hand, node mapping directly corresponds to RL actions. Note that node mapping positively influences the decisions of link mapping since the model for node mapping actions makes full use of both node features and link features (i.e., X and L in Eq. (2)), whereas a specific heuristic algorithm is used for link mapping. In our implementation, the shortest path algorithm is used.

Reward functions are defined subject to particular VNM objectives. For DCNet problems, rewards are calculated based on waiting times that affect the slowdown metric in Eq. (18),

$$\text{reward} = -|\mathcal{V}_{\text{wait}}| \times \Delta_{t,t-1} \quad (16)$$

where $\Delta_{t,t-1}$ denotes the time elapsed from the last mapping (i.e., the time interval between timesteps $t-1$ and t), and $|\mathcal{V}_{\text{wait}}|$ denotes the number of arriving virtual networks not yet allocated. For ISP problems, rewards are given consistently based on the acceptance ratio in Eq. (20),

$$\text{reward} = \begin{cases} 1 & \text{if } V \text{ is allocated} \\ -1 & \text{if failed} \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

For RL training, we use 2500 episodes where each is organized as follows. In Algorithm 5, a sample batch $\mathcal{B}$ containing 48 virtual networks ($\mathcal{V}$) and 1 substrate network ($\mathcal{S}$) is generated. Each episode is set to repeat $N = 16$ times. In DCNet

Algorithm 6 Run Episode Using **Gemma**

```

1: // Input: model parameters  $\theta$ , batch  $\mathcal{B}$ , rollout index  $i$ 
2: timestep  $t \leftarrow 0$ , reward samples  $R^i \leftarrow []$ , prob. samples  $\Pi^i \leftarrow []$ 
3: Update resource  $X_{\mathcal{C},\tau}^{(S)}$  using Eq. (11)
4: for sample  $V \in \mathcal{V}$  do
5:   mapping plan  $MAP \leftarrow []$ 
6:    $E^{(V)} \leftarrow \text{NetEmbed}(X^{(V)})$  using Eq. (2)
7:    $E_{\tau}^{(S)} \leftarrow \text{NetEmbed}(X^{(S)})$  using Eq. (2)
8:    $\mathcal{O}_k^{(V)} \leftarrow \mathcal{O}^{(V)}$ ,  $X_{bk}^{(S)} \leftarrow X^{(S)}$ 
9:   for  $j \leftarrow 1$  to  $|\mathcal{O}_k^{(V)}|$  do
10:     $E_{\text{new}}^{(S)} \leftarrow \text{NetEmbed}(X^{(S)})$  using Eq. (2)
11:    Update resource  $X^{(S)}$  using Eq. (11)
12:    if  $|\sum_{s \in \mathcal{O}_k^{(S)}} e_{\text{new}}^{(s)} - \sum_{s \in \mathcal{O}_k^{(S)}} e_{\tau}^{(s)}| \geq \theta$  then
13:       $E_{\tau}^{(S)} \leftarrow E_{\text{new}}^{(S)}$ ,  $X_{\mathcal{C},\tau}^{(S)} \leftarrow X_{\mathcal{C}}^{(S)}$ 
14:    end if
15:     $E^{(S)} \leftarrow E_{\tau}^{(S)}$ 
16:    Get  $(v', s')$  and prob.  $p$  using Alg. 3 for NodeMap
17:    //  $p \sim \pi_{\theta}(s_t^i = (X^{(V)}, X^{(S)}), a_t^i = (v', s'))$ 
18:    if  $(v', s')$  is NULL then
19:       $X^{(S)} \leftarrow X_{bk}^{(S)}$ ; break // unsuccessful mapping
20:    end if
21:    Allocate  $v'$  on  $s'$  in the environment
22:    Receive reward  $r_t^i$  using Eq. (16) or (17)
23:     $R^i \leftarrow R^i + [r_t^i]$ ,  $\Pi^i \leftarrow \Pi^i + [p]$ ,  $\mathcal{O}_k^{(V)} \leftarrow \mathcal{O}_k^{(V)} - \{v\}$ ,
24:     $t \leftarrow t + 1$ ,  $MAP \leftarrow MAP + [(v, s)]$ 
25:    Update resource  $X^{(S)}$  using Eq. (7)
26:  end for
27:  for  $(v, s) \in MAP$  do
28:    allocate virtual node  $v$  at substrate node  $s$ 
29:  end for
30:  for  $j \leftarrow |\Pi^i| - 1$  to  $1$  do
31:     $R_j^i \leftarrow R_j^i + R_{j+1}^i$ 
32:  end for
33: return  $R^i$ ,  $\Pi^i$  // return sequence

```

problems, virtual networks individually arrive online over time during the episode. When a virtual network is allocated to the substrate network, substrate resources are consumed; upon job completion by the virtual network, the resources are reclaimed. When all the virtual networks have been processed, the episode ends. While the model performs mapping over timesteps, the execution time passes. Specifically, we assume that over-utilization of link bandwidth affects the actual execution time of virtual networks that are allocated and processed, while the other finite resource types (e.g., CPU, memory) cannot be over-utilized according to the allocable test in Eq. (5).

In ISP problems, all the virtual networks are given as an offline batch job. Accordingly, unlike in DCNet, the actual execution time does not increase while the model performs mapping. In addition, the episode is set to end when the model cannot find a feasible allocation plan for unallocated virtual networks after more than 5 attempts.

Note that Algorithm 5 includes **NetEmbed** and it is required to simulate the behaviors of the **EMModel** model in Eq. (15) to apply the model-based selective embedding (in Algorithm 4) for **NetEmbed**. This is because the **EMModel** model might not be available when **Gemma** is being trained by Algorithm 5. That is, learning **EMModel** requires data sampled by **Gemma**. Thus, we use a simulation approach in which the substrate network embeddings are recalculated for each action, but they are reflected only in the case of relatively large changes (i.e., lines 14-16 in Algorithm 5).

V. EVALUATION

In this section, we compare the performance of **Gemma** under two previously described representative VNM problems, DCNet and ISP, with several other heuristic and learning-based methods.

Gemma is implemented in Python v3.6 and PyTorch v1.2 [22]. We train and test it on a system of an Intel(R) Core(TM) i9-9940X processor with an NVIDIA RTX 2080 GPU. Table 1 lists the hyperparameter settings for model training, using the RMSProp optimizer. We also implement a simulator for VNM system environments, that can interact with **Gemma**. We exploit different configurations to verify the behavior of **Gemma** through simulation.

TABLE 1. Hyperparameter setting.

	Description	Default Setting
Dataset	Num. of training samples	8000
	Num. of testing samples	1000
NeuralNet Setting	Num. of layers in MLP	2
	Embedding dimension (d)	12
	Decoder parameter	8, 1
	Activation	LeakyReLU
Model Fitting	Rollout size	16
	Learning rate	20^{-4}
	Num of epochs	1200
	Optimizer	RMSProp

To generate substrate network data samples for DCNet, we exploit the *fat-tree* [17] structure widely used in data center network research [1], [23]–[25]. To create the typology of virtual network data samples, we exploit the DAG-shaped query structure of the *TPC-H* benchmark [26], as was done in the data processing job structure in [8]. For ISP data samples, we exploit *Erdos-Renyi* random graphs [19] widely used in research of network virtualization [21]. Resource specifications on nodes and links are determined according to given probability distributions. Table 2 and 3 list the stochastic rules for DCNet and ISP data samples, respectively.

A. EVALUATION IN DCNet

To evaluate the VNM performance in DCNet problems where data-parallel jobs are formulated in a set of virtual networks $\mathcal{V}$, we use the slowdown metric [8], [27],

$$\text{slowdown} = \frac{1}{|\mathcal{V}|} \sum_{V \in \mathcal{V}} \frac{\text{Resp. time of } V}{\text{Exec. time of } V} \quad (18)$$

TABLE 2. Data generation rules for DCNet: (x_i, x_j) denotes a link between two nodes. N and U denote normal and uniform distributions, respectively. Req. denotes resource requests (or workloads) by virtual networks and Cap. denotes resource capability by substrate networks. Resource heterogeneity denotes a control parameter p such that the resource capacity is randomly set by $N(\mu, \sigma)$ with p probability and it is deterministically set by μ value with $1 - p$ probability; the larger p is, the slightly more heterogeneous the resource capacity is. In a substrate network S , the number of nodes $|\mathcal{O}^{(S)}| = 0.25k^3 + 1.25k^2$ where k denotes the pod size in a fat-tree topology, $0.25k^3$ corresponds to the number of computing nodes, and $1.25k^2$ corresponds to the number of switching nodes [17]. In our default setting of $k = 4$, we have $|\mathcal{O}^{(S)}| = 36$ for a substrate network with 16 computing nodes and 20 switching nodes.

Parameter	Generation Rule	Default
$x_{\text{comp}}^{(v)}$: comp. req. of $v \in V$	$N(\mu : 40 \sim 60, \sigma : 25)$	$\mu=50$
$x_{\text{mem}}^{(v)}$: mem. req. of $v \in V$	$N(\mu : 40 \sim 60, \sigma : 25)$	$\mu=50$
$(x_i^{(v)}, x_j^{(v)})_{\text{bw}}$: bandwidth req. in V	$N(\mu : 32 \sim 48, \sigma : 20)$	$\mu=40$
$x_{\text{comp}}^{(s)}$: comp. cap. of $s \in S$	$N(\mu : 120 \sim 180, \sigma : 50)$	$\mu=150$
$x_{\text{mem}}^{(s)}$: mem. cap. of $s \in S$	$N(\mu : 120 \sim 180, \sigma : 50)$	$\mu=150$
$(x_i^{(s)}, x_j^{(s)})_{\text{bw}}$: bandwidth cap. in S	$\{\mu, 2\mu, 4\mu\}$	$\mu=150$
$ \mathcal{O}^{(V)} $: num. of nodes in V	$U(2, 14)$	-
$ \mathcal{O}^{(S)} $: num. of nodes in S	$\{36, 99, 208\}$	36
Execution time of $v \in V$	$\{3, 4.5, 6, \dots, 13.5\}$	-
Num. of virtual networks V	48	-
Arrival rate of virtual networks V	$\frac{1}{\exp(5 \sim 15)}$	$\mu=0.1$
Resource heterogeneity p of $s \in S$	$\{0.3, 0.4, \dots, 0.7\}$	0.5

TABLE 3. Data generation rules for ISP: the resource and workload features for nodes in ISP are determined in the same way as in Table 2 for DCNet. However, the features for links are determined differently. The ISP topology, which is based on *Erdos-Renyi* random graphs [19], is structured differently from that of the DCNet fat-tree. In ISP, link probability denotes the probability that a pair of nodes connects.

Parameter	Generation Rule	default
$(x_i^{(v)}, x_j^{(v)})_{\text{bw}}$: bandwidth req. in V	$U(\mu - 10, \mu + 10)$	$\mu = 20$
$(x_i^{(s)}, x_j^{(s)})_{\text{bw}}$: bandwidth. cap. in S	$U(\mu - 20, \mu + 20)$	$\mu = 70$
$ \mathcal{O}^{(V)} $: num. of nodes in V	$U(2, 20)$	-
$ \mathcal{O}^{(S)} $: num. of nodes in S	$\{40, 45, \dots, 60\}$	50
Num. of virtual networks V	25	-
Link probability of V	0.25	-
Link probability of S	$\{.28, .36, .44, .52, .6\}$	.44

where the execution (Exec.) time is given as part of V 's specification and the response (Resp.) time is the total elapsed time from V 's arrival to completion. In DCNet, we presume that virtual network requests arrive individually over time, and the specifications of each, such as resource requirements and execution time, are known upon their arrival at runtime.

The slowdown metric represents the substrate network performance in terms of the delay to complete tasks that correspond to a set of virtual network requests. To obtain the response time of virtual network V , in addition to the waiting time before V is allocated, our environment simulates the actual processing time of V , which is influenced by the condition of the substrate network links.

As the link resources can be shared by several virtual networks, the environment uses the provisioning link ratio ($pr_i^{(V)}$ in Eq. (19)) on the substrate network to approximate the actual processing time. At the current timestep t , the provisioning link ratio is calculated, and the remaining execution time of V decreases according to the expected load that can be completed in the period spanning timesteps

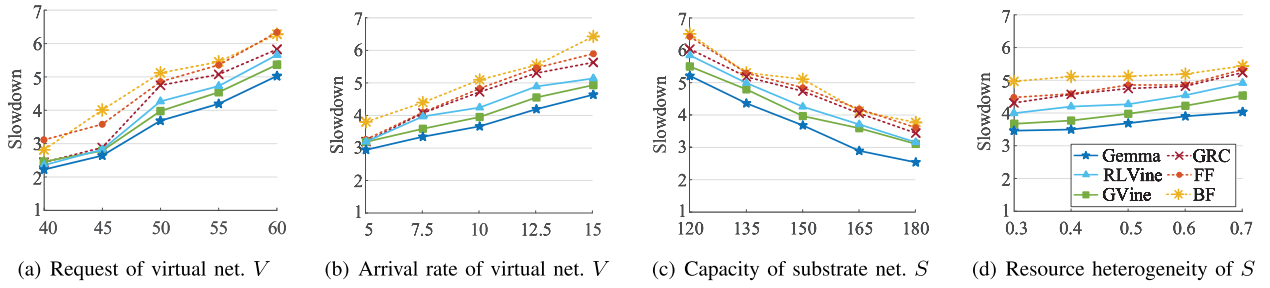


FIGURE 4. VNM performance for DCNet with respect to workload and resource conditions: the X-axis of the graphs denotes (a) the mean resource request of each node v in virtual networks V , (b) the mean arrival time of virtual networks, (c) the mean resource capacity of each node s in substrate network S , and (d) the resource heterogeneity of each node s in substrate network S . The distributions and specifications for those workload and resource conditions are described in Table 2.

$t - 1$ to t . This is given as

$$\begin{aligned}
 ptime_t^{(V)} &= \Delta_{t,t(V)} + rtime_t^{(V)} \\
 rtime_t^{(V)} &= (rtime_{t-1}^{(V)} - \Delta_{t,t-1}) \times \frac{pr_t^{(V)}}{pr_{t-1}^{(V)}} \\
 pr_t^{(V)} &= \alpha \max \left(\max_{(x_i^{(s)}, x_j^{(s)}) \in \mathcal{E}^{(V)}} \left(\frac{(x_i^{(s)}, x_j^{(s)})_{bw.*}}{(x_i^{(s)}, x_j^{(s)})_{bw.}} \right), 1 \right)
 \end{aligned} \quad (19)$$

where $ptime_t^{(V)}$ denotes the processing time of V until the timestep t , $rtime_t^{(V)}$ denotes the remaining execution time of V , and $\mathcal{E}^{(V)}$ is the set of links (edges) of the substrate network S allocated for V . Note that $(x_i^{(s)}, x_j^{(s)})_{bw.*}$ and $(x_i^{(s)}, x_j^{(s)})_{bw.}$ denote the allocated (used) link capacity and the initial (given) link capacity between s_i and s_j respectively. In addition, $\Delta_{t,t-1}$ denotes the elapsed time between two subsequent timesteps, $\Delta_{t,t(V)}$ denotes the time elapsed from V 's allocation to t , and α is a scale parameter. Notice that when $rtime_t^{(V)} = 0$, the actual processing time to complete V is $ptime_t^{(V)}$. This allows us to obtain the response time of V and the slowdown in Eq. (18).

For comparison purposes, we implement and test several other algorithms applicable to DCNet problems, in addition to **Gemma**.

- GRC [7] is a heuristic algorithm based on global capacity resource (GRC) which calculates the ranks of virtual and substrate nodes separately to exploit the integrated features of node and link capacities. Its rank calculation is intended to minimize the available node capacity while maximizing the available link capacity. Given calculated ranks, the algorithm assigns the highest-ranking virtual node to the highest-ranking substrate node.
- FF and BF are heuristic algorithms exploiting the first-fit and best-fit strategies for node mapping, respectively.
- RLVINE is a state-of-the-art learning-based VNM algorithm introduced in [28]. It uses a one dimensional convolution policy network for node mapping decisions trained by the Q-learning algorithm.

- GVine is a state-of-the-art learning-based VNM algorithm, GraphVine [29]. This algorithm uses a generative adversarial network model to perform substrate network embedding and K-means-based clustering to aggregate virtual nodes, thereby establishing an efficient node mapping process.

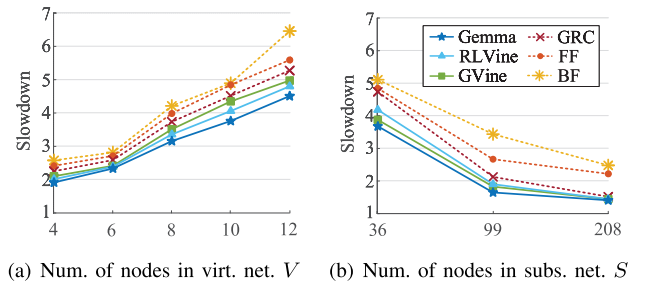


FIGURE 5. VNM performance for DCNet with respect to network scales: the X-axis of the graphs denotes (a) the mean number of nodes in virtual networks V and (b) the mean number of nodes in substrate networks S .

Figure 4 represents the VNM performance in slowdown for DCNet with respect to workload and resource conditions. As observed, **Gemma** outperforms the other methods. In Figure 4(a), the larger are the resource demands by virtual networks, the greater is the substrate network workload, and thus the larger is the slowdown it achieves. As shown in Figure 4(b), high arrival rates by virtual networks produce the same result. In Figure 4(c), substrate nodes with a large resource capacity produce the inverse result by providing more available resources. Additionally, as shown in Figure 4(d), the more severe resource heterogeneity of substrate nodes tends to allow greater fragmentation, which imposes more restrictions on node mapping in most cases. Most importantly, in most cases, **Gemma** maintains lower (better) slowdowns than the other methods including the learning-based algorithms, i.e., RLVine and GVine. It establishes a robust model in various VNM workload and resource conditions by employing both graph embedding and RL techniques.

Furthermore, we evaluate the VNM performance with respect to network scales. In Figure 5(a), large virtual

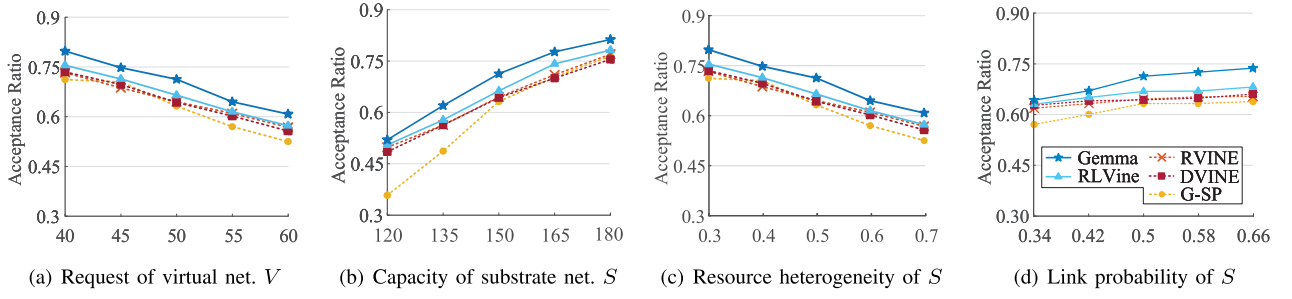


FIGURE 6. VNM performance for ISP with respect to workloads and resource conditions: the X-axis of the graphs denotes (a) the mean resource request of each node v in virtual networks V , (b) the mean resource capacity of each node s in substrate network S , (c) the resource heterogeneity of each node s in S , and (d) the link probability of S . The distributions and specifications for the workload and resource conditions are described in Table 2 and 3.

networks demand more resources, increasing the slowdown. Conversely, in Figure 5(b), larger substrate networks have more nodes and thus greater resource availability, which decrease the slowdown. For both cases on various scales, **Gemma** consistently yields better performance than the other heuristic and learning-based algorithms we tested. For instance, it yields 2.3~7.4% improvements over RLVine and 3.3~13.5% over GVine with respect to the virtual network scales in Figure 5(a).

B. EVALUATION IN ISP

To evaluate the VNM performance in ISP problems in which a batch of network packet processing jobs $\mathcal{V}$ is specified, we use the acceptance ratio metric,

$$\text{Accept. ratio} = \frac{\text{Num. of allocated } V \in \mathcal{V}}{|\mathcal{V}|}. \quad (20)$$

This metric is commonly used in the context of resource management in network virtualization [1], [13].

We compare several algorithms applicable to ISP with **Gemma**.

- **RVINE** and **DVINE** [30] are heuristic algorithms. Using distance information on a substrate network, they generate virtual substrate nodes. Each virtual node is assigned to one of the virtual substrate nodes, and reassigned to a substrate node depending on occupied link resources. For reassignment, the DVINE algorithm selects the substrate node with the largest number of link resources in a deterministic manner, while the RVINE algorithm utilizes the randomized rounding technique based on the probability distribution for the occupied link resources.
- **G-SP** [31] is a greedy node mapping algorithm in which a virtual node is assigned to the substrate node with the largest expected revenue. The product of the node and link resource features for each substrate node is used to calculate the revenue.
- **RLVINE** denotes the previously explained learning-based algorithm.

Figure 6 shows that **Gemma** outperforms the other methods in terms of the acceptance ratio in Eq. (20) for ISP problems with respect to various workload and resource conditions. Unlike in DCNet in which virtual network requests

continuously arrive and are processed online, VNM for ISP addresses offline batch scheduling upon a set of virtual networks. In this ISP problem setting, **Gemma** demonstrates its robustness, showing higher acceptance ratios than the other algorithms tested.

Furthermore, we also consider the case where offline batch scheduling is conducted with existing mapping in response to changes of substrate network capacity, similar to [15], [32]. For virtual node v allocated to substrate node s_v , this case might introduce a reconfiguration cost to move v from s_v to another substrate node s'_v . We incorporate the reconfiguration cost into the reward design in Eq (17) as

$$1 - \beta \sum_{v \in \mathcal{O}(V)} \mathcal{L}(s_v, s'_v) \quad \text{if } V \text{ is allocated} \quad (21)$$

where $\mathcal{L}(s_v, s'_v)$ calculates the relocation cost to move v from s_v to s'_v and β is a relocation cost weight. In our implementation, we exploit the distance from the existing mapped substrate node s_v to the newly mapped node s'_v to calculate $\mathcal{L}(s_v, s'_v)$, which is weighted by β . While other factors, e.g., downtime, can be used to estimate the relocation cost $\mathcal{L}$, we focus on the general structure of **Gemma** in which the reward can be defined according to an application-specific optimization criterion, as in Eq. (21).

Figure 7 shows the performance in acceptance ratio and the relocation ratio with respect to various relocation cost weight values β . As shown in Figure 7(b), as β increases, the number of relocated virtual nodes decreases; this accounts for the slight degradation in acceptance ratio for larger β in Figure 7(a), as reconfiguration overheads act as an additional constraint when mapping between virtual and substrate nodes. In this reconfiguration test, we update the resource capacity of substrate network S between two successive rounds of offline batch mapping; for each resource type c , its capacity for substrate node s is changed from the previous value $x_{c,pre}^{(s)}$ to the new value $x_{c,cur}^{(s)}$ as

$$x_{c,cur}^{(s)} = x_{c,pre}^{(s)} + N\left(\mu : 0, \sigma : \epsilon x_{c,pre}^{(s)}\right) \quad \text{for } c \in \mathcal{C}, s \in \mathcal{O}^{(S)} \quad (22)$$

where ϵ is a change weight.

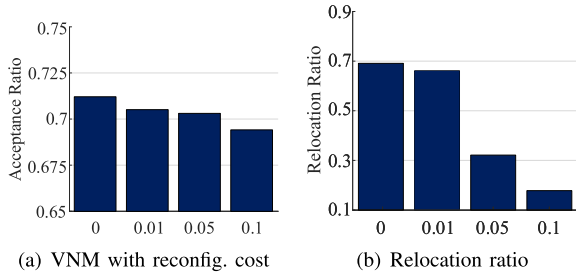


FIGURE 7. VNM reconfiguration in ISP problem settings: the X-axis of the graphs denotes the reconfiguration cost weight β in Eq. (21). In (a), the Y-axis denotes the acceptance ratio as usual; accordingly, the case of $\beta = 0$ corresponds to the default case of $X = 50$ in Figure 6(a). In (b), the Y-axis denotes the relocation ratio, i.e., $\frac{\text{num. of relocated virtual nodes}}{\text{num. of virtual nodes}}$.

C. EFFECT OF OPTIMIZATION SCHEMES

The **Gemma** framework is based on the RL-based encoder-decoder model architecture that incorporates two optimization schemes. We analyze the effects of the optimization schemes by comparing several variants of **Gemma**.

- **G-Plain** is a plain encoder-decoder model. This implementation makes use of the graph convolutional network (GCN) encoder in [21]. To incorporate various substrate network typologies in a single model, unlike [21], however, it additionally performs iterative ranking operations through the decoder.
- **G-Model** is an encoder-decoder model that uses the model-based selective embedding scheme.
- **G-2Stage** is an encoder-decoder model that uses the two-stage mapping scheme.
- **Gemma** is our proposed **Gemma** model that uses both optimization schemes.

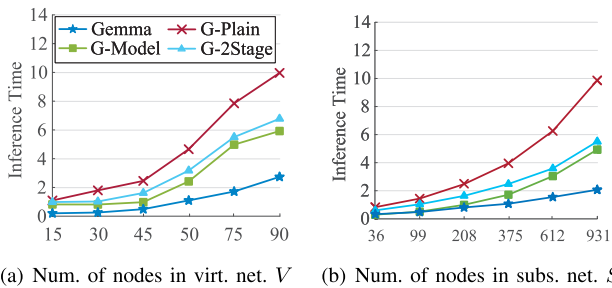


FIGURE 8. Gemma variants' inference time for DCNet with respect to network scales: the X-axis of each graph denotes (a) the mean number of nodes in virtual networks V and (b) the mean number of nodes in substrate network S . The Y-axis denotes the inference time in seconds.

In Figure 8, we show the effect of the optimization schemes in terms of the achieved inference time for DCNet. The two optimization schemes reduce the inference time with greater reduction as the network scale increases. Specifically, in Figure 8(a), **Gemma** maintains its low inference time for large virtual networks, no longer than 3 seconds, unlike the other **Gemma** variants. In Figure 8(b), a similar pattern emerges for large substrate networks. It is important to achieve such low model inference times, especially when

handling online scheduling or node mapping problems such as DCNet.

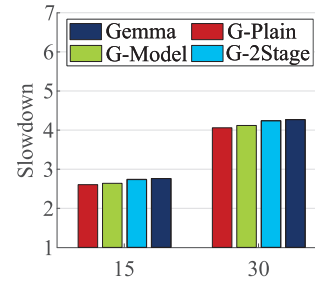


FIGURE 9. Gemma variants' performance for DCNet: the X-axis denotes the mean number of nodes in virtual networks V .

We also compare the **Gemma** variants in terms of their model performance in Figure 9. As observed, the optimization schemes tend to slightly degrade the slowdown, but by no more than 5% compared to **G-plain** which intensively computes the mapping scores of all candidate node pairs. In general, those changes are insignificant considering the gain in the observed inference times illustrated in Figure 8, particularly for mapping large virtual networks online. The results in Figure 8 and 9 clearly indicate the benefits of **Gemma**'s optimization schemes: they reduce the inference time without significantly reducing the model accuracy.

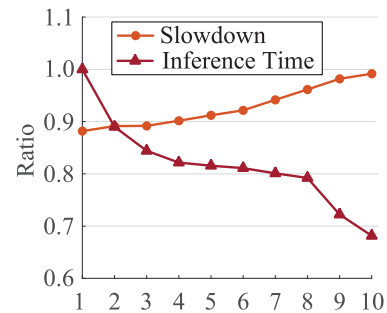


FIGURE 10. Model-based selective embedding: the X-axis denotes the threshold θ of Algorithm 4. The worst slowdown is standardized to one and the best inference time is standardized to one on the Y-axis for trade-off representation.

In **Gemma**, the model-based selective embedding scheme has a threshold that can control the trade-off between VNM performance and inference time. Figure 10 represents the trade-off between the achieved inference times and slowdowns for DCNet, which depends on the threshold values. Larger threshold values facilitate greater use of previous embeddings of time-varying substrate networks, producing a shorter inference time but a larger slowdown. In our implementation, we empirically set the threshold to $\theta = 3$ in Algorithm 4.

VI. RELATED WORKS

Network virtualization has been used as a fundamental technology to enable the physical resources of a substrate

network to be shared among multiple virtual networks, enhancing the architecture and usability of the Internet and modern cloud computing centers. Virtual network mapping (VNM) or embedding is a key component of network virtualization that is responsible for allocating substrate network resources to virtual network elements. Numerous heuristic algorithms have been proposed to provide optimal or profitable allocation, given several assumptions about VNM objectives, topologies and resource properties of substrate networks, and workload patterns of virtual networks [1]–[6], [23], [24].

In recent years, learning-based approaches have been well-received as a common enabling technology to handle various VNM problems. In DeepVine [11], online VNE decisions were formulated as reinforcement learning (RL) problems where an RL agent continuously determines the mapping for arriving virtual networks. DeepVine leverages the automatic feature extraction capability of convolutional neural networks by representing the resource properties of substrate and virtual networks in the form of images. Similarly, in [12], a Q-learning-based VNM solution was presented. It uses Q-values calculated for substrate and virtual nodes to find the optimal mapping strategy for data center networks. Those approaches revealed the benefits of employing RL methods to solve VNM problems. Our work also leverages deep learning models driven by RL, but it concentrates on the encoder-decoder neural network architecture with graph embedding. This encoder-decoder architecture can handle substrate networks with various typologies and resource capacities in a single trained model. Similarly, the recent work in [28] supports the use of iterative ranking procedures in substrate networks of various sizes. Unlike our work leveraging graph convolutional networks (GCNs) for embedding, it uses a simple one-dimensional convolution process. Thus, its internal representation might not fully reflect the structure of complex networks.

Graph neural networks are used to learn to extract features from complex data described as graphs. Because of the structural similarity between graph data and communication network topologies where nodes and edges exist, graph neural networks are used to adapt deep learning techniques for VNM problems. In [29], GraphVine exploited a GCN-based autoencoder to represent complex graph data in a latent space and cluster similar nodes in a substrate network through the latent space. A cluster is selected for assigning the nodes of a selected virtual network. However, that model exploits heuristics in the final mapping phase, which decreases its performance when constraints require that virtual nodes be mapped to other clusters. The work in [21] also leveraged graph neural networks. Specifically, it adopts GCN and RL, similar to our work. However, its model implementation requires retraining for every substrate graph typology due to its specific learning structure, which uses a fully connected layer for the given substrate network.

VII. CONCLUSION

In this paper, we have presented **Gemma**, an RL-driven graph embedding and mapping framework for addressing various virtual network mapping (VNM) problems. To improve its runtime and performance in large-scale networks, we use an encoder-decoder deep learning architecture with two optimization schemes, two-stage mapping and model-based selective embedding. Through experiments with two representative VNM scenarios that use various topological and workload conditions, we have shown that **Gemma** outperforms other heuristic and learning-based approaches.

The direction of our future work is to adapt the graph embedding capability in **Gemma** to a distributed model serving system in which a set of machine learning models is operated in a network of resource-constrained embedded devices.

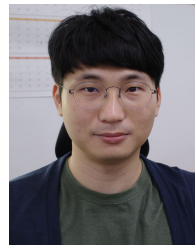
REFERENCES

- [1] S. Haeri and L. Trajkovic, "Virtual network embeddings in data center networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 874–877.
- [2] T. Wang and M. Hamdi, "Presto: Towards efficient online virtual network embedding in virtualized cloud data centers," *Comput. Netw.*, vol. 106, pp. 196–208, Sep. 2016.
- [3] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer, "Energy efficient virtual network embedding," *IEEE Commun. Lett.*, vol. 16, no. 5, pp. 756–759, May 2012.
- [4] K. D. Chinmaya and K. S. Prasan, "DYVINE: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1029–1045, May 2019.
- [5] W. Wei, H. Gu, K. Wang, X. Yu, and X. Liu, "Improving cloud-based IoT services through virtual network embedding in elastic optical inter-DC networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 986–996, Feb. 2019.
- [6] Y. Liu, P. Han, J. Hou, and J. Zheng, "Resource-efficiently survivable IoT services provisioning via virtual network embedding in fiber-wireless access network," *IEEE Access*, vol. 7, pp. 65007–65018, 2019.
- [7] S. Haeri, Q. Ding, Z. Li, and L. Trajkovic, "Global resource capacity algorithm with path splitting for virtual network embedding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 666–669.
- [8] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 270–288.
- [9] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, vol. 8, pp. 120388–120416, 2020.
- [10] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 106–120, Mar. 2017.
- [11] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 879–885.
- [12] Y. Yuan, Z. Tian, C. Wang, F. Zheng, and Y. Lv, "A Q-learning-based approach for virtual network embedding in data center," *Neural Comput. Appl.*, vol. 32, no. 7, pp. 1995–2004, Apr. 2020.
- [13] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.
- [14] P. Han, L. Guo, Y. Liu, X. Wei, J. Hou, and X. Han, "A new virtual network embedding framework based on QoS satisfaction and network reconfiguration for fiber-wireless access network," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.
- [15] V. Eramo, E. Miucci, and M. Ammar, "Study of reconfiguration cost and energy aware VNE policies in cycle-stationary traffic scenarios," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1281–1297, Apr. 2016.

- [16] Z. Zhang, S. Su, J. Zhang, K. Shuang, and P. Xu, "Energy aware virtual network embedding with dynamic demands," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5386–5391.
- [17] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.
- [18] Y. Li and Y. Zhang, "EPVNE: An efficient parallelizable virtual network embedding algorithm," *Wireless Commun. Mobile Comput.*, vol. 2019, pp. 1–10, Nov. 2019.
- [19] E. N. Gilbert, "Random graphs," *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, 1959.
- [20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–13.
- [21] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.
- [22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
- [23] J. Duan, Z. Guo, and Y. Yang, "Embedding nonblocking multicast virtual networks in fat-tree data centers," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2015, pp. 123–132.
- [24] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A comparative analysis of data center network architectures," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 3106–3111.
- [25] C. K. Dehury and P. K. Sahoo, "Failure aware semi-centralized virtual network embedding in cloud computing fat-tree data center networks," *IEEE Trans. Cloud Comput.*, early access, Apr. 2, 2020, doi: [10.1109/TCC.2020.2984604](https://doi.org/10.1109/TCC.2020.2984604).
- [26] T. Chiba and T. Onodera, "Workload characterization and optimization of TPC-H queries on apache spark," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2016, pp. 112–121.
- [27] M. Cheong, H. Lee, I. Yeom, and H. Woo, "SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153432–153444, 2019.
- [28] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.
- [29] F. Habibi, M. Dolati, A. Khonsari, and M. Ghaderi, "Accelerating virtual network embedding with graph neural networks," in *Proc. 16th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2020, pp. 1–9.
- [30] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.
- [31] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.
- [32] V. Eramo, E. Miucci, and M. Ammar, "Study of migration policies in energy-aware virtual router networks," *IEEE Commun. Lett.*, vol. 18, no. 11, pp. 1919–1922, Nov. 2014.



MINJAE PARK is currently pursuing the degree with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. He plans to continue his studies at graduate school. His research interests include intelligent application, reinforcement learning, and network systems.



Foundation, from 2013 to 2017.

YOUNGSEOK LEE received the B.S. degree in electrical and electronic engineering from Sungkyunkwan University, Suwon, South Korea, in 2013, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. His research interests include intelligent application, software engineering, network traffic analysis, and wireless networks. He was a recipient of the Global Ph.D. Fellowship by the Korea National Research



the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. His research interests include AQM, congestion control, TCP, wireless networks, and future internet architecture.

IKJUN YEOM received the B.S. degree in electronic engineering from Yonsei University, Seoul, South Korea, in February 1995, and the M.S. and Ph.D. degrees in computer engineering from Texas A&M University, in August 1998 and May 2001, respectively. He worked at DACOM Company, from 1995 to 1996, and Nortel Networks, in 2000. He had been an Associate Professor with the Department of Computer Science, KAIST, from 2002 to 2008. He is currently a Full Professor with



ing, Sungkyunkwan University, Suwon, South Korea. His research interests include intelligent application, analytic monitoring, cloud computing, and networked cyber-physical systems.

HONGUK WOO (Member, IEEE) received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he worked with Samsung Electronics, Samsung Research, as a Principal Engineer and a Vice President. Since 2018, he has been an Assistant Professor with the Department of Computer Science and Engineering

...