IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

INNOVATIVE TRENDS IN 6G ECOSYSTEMS

Received 11 December 2023, accepted 19 December 2023, date of publication 22 December 2023,
date of current version 10 January 2024.

Digital Object Identifier 10.1109/ACCESS.2023.3346187

RESEARCH ARTICLE

An Autonomous Deployment Mechanism for AI Security Services

WEILIN WANG, HUACHUN ZHOU, MAN LI, AND JINGFU YAN
School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Huachun Zhou (hchzhou@bjtu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFA0701604, in part by NSFC under Grant 62341102, and in part by the Fundamental Research Funds for the Central Universities under Grant 2022YJS130.

ABSTRACT Future network architectures are expected to be autonomous, intelligent, and service-based, posing new security challenges. To address these challenges, the Artificial Intelligence (AI) security service emerges as a promising solution. However, the complex service configurations and performance guarantees hinder the autonomous deployment of the AI security service. This paper proposes an autonomous deployment mechanism in Software-Defined Networking/Network Function Virtualization (SDN/NFV) enabled networks. First, our mechanism introduces user and decision planes on top of the control plane, enabling hierarchical intent expression and translation from user security intent to security policies. Then, we analyze the embedding problem of the AI-based Security Function Chain (AISFC) during security policy generation. We formulate the AISFC embedding problem as an Integer Linear Programming (ILP) task to minimize the total response delay. By decomposing it into AISF placement and routing, we design a heuristic algorithm with polynomial time complexity. Finally, we validate the proposed mechanism through a prototype system and numerical simulations, demonstrating its ability to autonomously translate, implement, and guarantee the user security intent. Comparative analysis shows that our approach considering the relationship between available computing resources and delay achieves smaller response delays than the baseline. Furthermore, our algorithm achieves a gap from optimality approximately 28.57% smaller than the greedy algorithm and supports networks that are 4.34 times larger in scale than the exact solution within a 2-second execution time.

INDEX TERMS Artificial intelligence, network function virtualization, service function chain, intent-based network, security management.

I. INTRODUCTION

Future networks are expected to support diverse and differentiated service scenarios [1], including autonomous driving, telemedicine, and smart cities. To achieve adaptability across various scenarios, Service-Based Architecture (SBA) [2] leveraging software and virtualization technology is crucial. In the forthcoming 6G era, the integration of Artificial Intelligence (AI) technology will enable an autonomous and intelligent SBA, facilitating ubiquitous AI as a Service by autonomously managing and allocating resources and functions in cloud-edge-end networks [3], [4], [5], [6].

The evolution of network architecture brings forth new security challenges. The emergence of human-computer-object interaction scenarios in 6G systems blurs network boundaries, increasing the severity of security threats [7], [8]. Traditional protection methods at network boundaries, such as access control and attack detection, face limitations, necessitating new end-to-end security detection and monitoring mechanisms. Moreover, under the SBA, on-demand security services are crucial, as different application scenarios require the network to dynamically provide customized security capabilities to users.

Service Function Chain (SFC) [9] based on Software-Defined Networking (SDN) and Network Function Virtualization (NFV) technology offers logically independent network function paths within a shared virtualized

The associate editor coordinating the review of this manuscript and approving it for publication was Zaharias D. Zaharis.

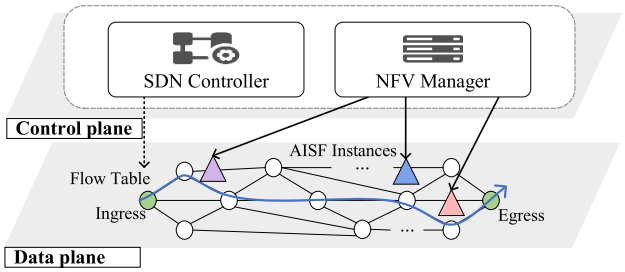


FIGURE 1. An example of implementing AI security services in the SDN/NFV-enabled network.

infrastructure, making it an excellent candidate for implementing end-to-end security services. The integration of AI technology in network security has given rise to AI-based Security Functions (AISFs) [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. These security functions utilize AI techniques to analyze network data, model normal and attack behaviors, and perform user authentication, attack detection, mitigation, source tracing, traffic classification, etc. AISFs can be deployed and orchestrated through SFC, enabling traffic to traverse different security paths and providing customized security services to users. In the subsequent description, we refer to the SFC comprising AISFs as the AI-based Security Function Chain (AISFC), and the security service enabled by AISFC as the AI security service.

To illustrate the implementation of AI security services in an SDN/NFV-enabled network, FIGURE 1 depicts a representative example. The data plane includes ingress/egress nodes (green nodes) and function nodes (white nodes). Ingress/egress nodes are responsible for traffic encapsulation/decapsulation and forwarding based on the Network Service Header (NSH) [22] protocol, while AISF instances can be deployed on the function nodes. The control plane consists of the NFV manager and SDN controller, responsible for the instantiation, deployment, and orchestration of AISFs within the data plane.

Despite advancements, realizing the autonomous deployment of AI security services still poses unresolved challenges. This paper focuses on two specific issues in this regard.

- Firstly, the complexity of service configuration and the lack of user-friendliness hinder efficient deployment. Users often lack understanding of the technical details underlying the infrastructure and articulate their intent solely in terms of desired outcomes, rather than the necessary actions. This necessitates the network's ability to comprehend user intent, convert it into network-executable policies, and autonomously deploy and implement functions that align with the user's intent. Intent-Based Networking (IBN) has emerged as a promising approach to address this problem. Intent-driven security management [23], [24], [25], [26], [27], [28], [29] has received considerable attention from researchers. However, existing research primarily focuses on configuring specific security functions or operational security during service orchestration, overlooking the application of SFC-enabled security services.
- Secondly, there is a lack of suitable methods to ensure optimal performance of AISFs within the SFC. The performance of AISFs, which can be evaluated using security metrics such as detection rate, response delay, accuracy, and authentication efficiency, depends on the quality and complexity of the AISF model as well as the allocated network resources. Once the AISF model is determined, the allocation of network resources to each AISF within the AISFC becomes the key to influencing its response delay or efficiency. To ensure prompt response to security events, minimizing the response delay of AISF is crucial. The AISF with higher quality tends to have greater complexity and workload [30]. In resource-constrained environments such as edge computing [31], the varying computing resources across nodes significantly impact the computing delay of AISFs with heavier workloads. Existing research [32], [33], [34], [35], [36], [37] on SFC embedding problems often neglects the relationship between available computing resources and the computing delay of Virtualized Network Functions (VNF), which poses challenges in deploying AISFs in resource-constrained environments. To address these challenges and ensure optimal AISF performance, it is imperative to re-model the AISFC embedding problem and devise an effective resource scheduling method.

In this paper, we present an autonomous deployment mechanism for AI security services based on IBN. The proposed mechanism translates user security intent into the security capability requirement, generates network-executable AISFC and AISFC embedding policies, and ensures the autonomous implementation of AI security services that meet the user's security level requirement.

The main contributions of this paper are as follows:

- We design an autonomous deployment mechanism that introduces user and decision planes on top of the control plane. This mechanism establishes a scalable security capability model, security intent mapping graph, and knowledge base in the user and decision planes. It enables hierarchical expression and translation of the user's security intent, and security capability requirement into security policies. Through the collaboration of modules in the user plane, decision plane, and control plane, this mechanism provides AI security services that meet the user's security level requirement.
- We analyze the AISFC embedding problem in the process of generating the AISFC embedding policy. (i) We formulate the AISFC embedding problem as an Integer Linear Programming (ILP) task to minimize the total response delay. (ii) We propose the $\mathrm{ILP}^{\mathrm{C}}_{\mathrm{AISFC}}$ and baseline $\mathrm{ILP}^{\mathrm{C-assu}}_{\mathrm{AISFC}}$. $\mathrm{ILP}^{\mathrm{C}}_{\mathrm{AISFC}}$ considers the relationship between available computing resources and computing delay, while baseline $\mathrm{ILP}^{\mathrm{C-assu}}_{\mathrm{AISFC}}$ assumes the computing resource requirement for AISFs and considers any deployment meeting the requirement as acceptable in terms of computing delay.

(iii) We decompose the AISFC embedding problem into AISF placement and routing, design an AISF Placement and Routing with Rank and Fallback (APR^2F) algorithm, and prove its polynomial time complexity. The above model and algorithm optimize resource allocation in the AISFC embedding process, ensuring optimal AISF performance within the AISFC.

- We verify and evaluate the proposed mechanism, model, and algorithm through a prototype system and numerical simulations. Experimental results demonstrate that our mechanism achieves autonomous translation, implementation, and guarantee of user security intent. Compared to the baseline ILP^{C-assu}_{AISFC}, ILP^{C}_{AISFC} achieves smaller response delays especially in environments with significant differences in computing resources. The APR^2F algorithm provides solutions closer to optimality compared to the greedy algorithm and supports significantly larger-scale networks than the exact solution while maintaining the same execution time.

The rest of this paper is organized as follows: Section II summarizes related work; Section III designs the autonomous deployment mechanism for AI security services; Section IV analyzes the AISFC embedding problem; Section V shows the experimental results and analysis; Section VI summarizes the work of this paper.

II. RELATED WORK

This section summarizes related work from two aspects: security service automation and AISFC embedding.

A. SECURITY SERVICE AUTOMATION

Researchers have designed various AISFs for network applications, demonstrating their effectiveness and accuracy. For example, Zolanvari et al. [10] proposed **intrusion detection functions** based on machine learning techniques like random forests and decision trees, effectively detecting attacks such as Backdoor, Command Injection, and SQL Injection. Bhardwaj et al. [11] combined autoencoders and Deep Neural Networks (DNN) for flexible and accurate DDoS attack detection. Li et al. [12] used statistical methods and Convolutional Neural Networks (CNN) to tackle different types of DDoS attacks. Li et al. [13] designed a DDoS attack mitigation function based on federated learning for the industrial Internet of Things to effectively reduce the mitigation time. **Access control functions** have also received attention, with Picard and Pierre [14] developing a risk-based assessment system that dynamically authenticates users using deep reinforcement learning. Fu et al. [15] proposed a neural support decision tree-based approach to proactively evaluate and authenticate access requests. Fang et al. [16] designed a method to dynamically update trust relationships, providing access authorization that adheres to the zero-trust principle. Yang et al. [17] employed a relation network and behavior library to develop a security control function using deep learning techniques. They utilized specific information for control identification, ensuring privacy and security. Sinha et al. [18] introduced a low-overhead machine learning method for accurate DDoS **attack path tracking**. Li et al. [19]introduced a graph convolutional network-based multi-domain DDoS path tracing function. By leveraging graph structures, their approach accurately reconstructs DDoS attack paths. Additionally, scholars [20], [21] have studied AI-based **encrypted traffic classification** to ensure service quality. These diverse AISFs hold promise for building a more trustworthy network.

However, existing research primarily focuses on designing and evaluating individual security functions, without considering their deployment in real-world networks. Customized security services often necessitate the integration of multiple AISFs, resulting in complex and challenging deployment processes. Intent-driven autonomous network configuration has emerged as a relevant topic among scholars. User intent refers to operational goals and outcomes defined declaratively, without specifying implementation details [23]. The objective is to convert abstract user intents into network-executable policies and automate the network policy configuration process [24]. While some solutions have been proposed, such as the work by Chowdhary et al. [25] for efficient security policy management. However, this solution still requires users to possess prior knowledge of network security services. Szyrkowiec et al. [26] focused on intent-based security service automation but only considered encryption services at different network levels. Murcia et al. [27] utilized intent-based orchestration to tackle heterogeneity in distributed computing. However, their focus lies primarily on security aspects during the service request process rather than the request for a security service. Kim et al. [28] designed a security service automation provision system based on the Interface to Network Security Functions (I2NSF) architecture. Furthermore, Nguyen et al. [29] developed a conflict detection mechanism for I2NSF, effectively enhancing the architecture's robustness. However, they primarily addressed the configuration of specific security functions without considering the orchestration of security function paths.

Analyzing the existing research reveals a gap in the availability of an efficient and user-friendly autonomous platform/framework/mechanism for deploying customized AI security services.

B. AISFC EMBEDDING

In Section I, we discussed the importance of response delay in evaluating the impact of network resources on AISF performance. This subsection provides a summary of related work from SFC embedding with a delay guarantee and AI inference resource scheduling.

Several studies have addressed the SFC embedding problem with a delay guarantee. Yuan and Ren [32] proposed a multi-layer network design using a greedy heuristic algorithm to solve the SFC embedding problem with end-to-end delay constraints. Sasabe and Hara [33] introduced an enhanced network model and formulated the SFC embedding problem

as a capable ILP problem based on the shortest path roaming problem, proposing a heuristic algorithm for its solution. Li et al. [34] proposed a reinforcement learning-based method to address the SFC path selection problem without prior knowledge. Horimoto and Oki [35] tackle the limitation of existing methods, which solely focus on maximum latency, by introducing service delay probability protection. Varasteh et al. [36] and Wang et al. [37] both decomposed the SFC embedding problem into the VNF placement and routing problem. Paper [36] utilized a centrality-based ranking method for VNF-to-physical node mapping and an aggregation cost algorithm based on Lagrangian relaxation to find solutions satisfying the delay constraints. Paper [37] employed a multi-agent deep reinforcement learning framework for VNF placement and routing subtasks. It addressed the dynamic network topology by utilizing parameter transfer model retraining. However, these studies did not consider the relationship between available computing resources and VNF computing delay, rendering the proposed models or methods potentially unsuitable for AISFC deployment at the network edge.

AISFs can be categorized as AI inference tasks. The problem of AI inference resource scheduling has gained significant attention from researchers. Zhang et al. [30] proposed an edge AI model deployment framework that allows joint configuration of data quality ratio and model complexity ratio by analyzing the relationship between model quality, data, and model complexity. She et al. [38] addressed the on-demand scheduling of edge Deep Neural Network (DNN) inference tasks with guaranteed accuracy and delay, proposing an online heuristic algorithm to tackle the problem. Zhang et al. [39] employed a deep deterministic policy gradient-based learning algorithm to solve the task and resource allocation problem of DNN inference in the industrial Internet of Things (IoT) context. Ma et al. [40] devised a bandit learning-based scheduling scheme to ensure the reliability of DNN inference tasks. However, AISFs in the AISFC differ from the AI inference tasks discussed in the previous studies. Unlike those tasks, AISFs process traffic without relying on a pre-prepared dataset. Therefore, determining the response delay of AISFs requires considering the relationship between the processed samples and the traffic (or data packets).

Therefore, we aim to address the shortcomings of AISFC embedding in the aforementioned research. We propose an AISFC embedding model and method specifically tailored for resource-constrained environments.

III. AUTONOMOUS DEPLOYMENT MECHANISM

In this section, we propose an autonomous deployment mechanism for AI security services in SDN/NFV-enabled networks. It aims to autonomously translate user security intents into network executable policies and ensure the implementation of AI security services aligned with the user's security level requirement.

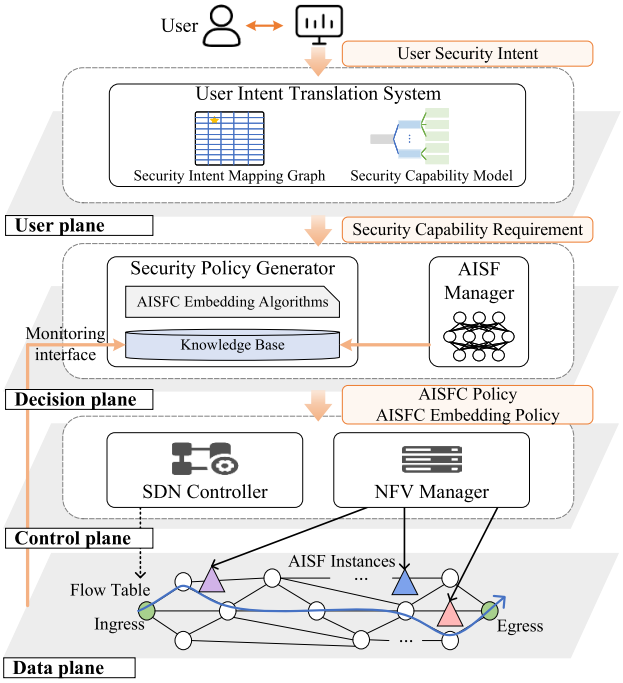


FIGURE 2. The autonomous deployment mechanism for AI security services.

A. OVERVIEW

The autonomous deployment mechanism for AI security services, depicted in FIGURE 2, extends the control plane in the SDN/NFV-enabled network by adding a user plane and a decision plane.

In the user plane, the user security intent is obtained through the Graphical User Interface (GUI) or Natural Language Processing (NLP) [41] system. It is then translated into the security capability requirement by the user intent translation system, using the security intent mapping graph and security capability model. The translated requirement is forwarded to the decision plane.

The decision plane consists of the security policy generator and the AISF manager. The security policy generator receives the security capability requirements and generates an AISFC policy based on the Security Capability Information (SCI) and AISF Information (AISFI) stored in the knowledge base. It also utilizes the AISFC embedding algorithm to create an AISFC embedding policy, considering the Substrate Network Information (SNI) in the knowledge base. The AISF manager evaluates AISF models provided by developers, saves qualified models, and updates the knowledge base with AISFI. The monitoring interface collects SNI periodically, monitors the inference results and running status of AISF instances, and makes timely adjustments to security policies to ensure continuous fulfillment of user intent.

To facilitate the transfer and configuration of information and policies between planes or function modules, we employ the YANG model to define the northbound interfaces between planes and the monitoring interface.

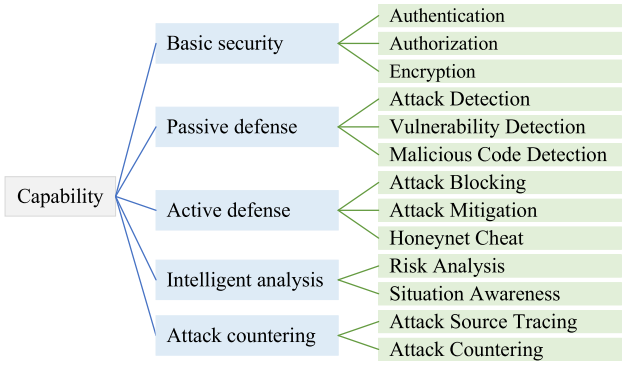


FIGURE 3. The security capability model.

Capability / Action	Basic security	Passive defense	Active defense	Intelligent analysis	Attack countering
Authenticate	★				
Authorize	★				
Detect		★			
Block		★	★		
Mitigate		★	★		
Analyze				★	
Guarantee	★	★	★	★	
Trace	★	★	★	★	★
Counter	★	★	★	★	★

FIGURE 4. The security intent mapping graph.

For more details, refer to the documentation available at https://github.com/wwl0220/AISSADM.

The following sections provide a detailed explanation of the function modules in the user plane and decision plane, as well as the hierarchical expression and translation process of user security intent between planes.

B. USER PLANE

1) USER SECURITY INTENT

User security intent encompasses two aspects. Firstly, it defines the required security capabilities, and secondly, it specifies the level of requirement for those capabilities. However, users often lack an understanding of the network's available security capabilities. User security intent typically represents a task that the network should perform, such as user identity authentication, attack detection, or protection of private information. We express a user security intent as follows:

$$UserSecurityIntent \Rightarrow (Iid, Conditions, Object, Action, SLR). \quad (1)$$

where *Iid* denotes the identification assigned by the translation system to the user security intent. *Conditions* denotes the execution conditions, such as network device range and time information. *Object* refers to the intent's target, *Action* represents the intended action, and *SLR* indicates the security level requirement. Based on the *Action*, user security intent can be categorized into actions like authenticating, authorizing, detecting, blocking, mitigating, analyzing, guaranteeing, tracing, and countering. *SLR* reflects the extent to which the user requires security capabilities.

2) SECURITY CAPABILITY MODEL AND SECURITY INTENT MAPPING GRAPH

Security capabilities, classified according to the Sliding Scale of Cyber Security's five stages [42], encompass basic security, passive defense, active defense, intelligent analysis, and attack counter. These capabilities are composed of various sub-capabilities, as depicted in FIGURE 3. The sub-capabilities can be further categorized based on the *Object*. For instance, attack detection can be subdivided into DDoS attack detection and Web attack detection.

FIGURE 4 depicts the security intent mapping graph. Intents such as authenticating, authorizing, detecting, blocking, mitigating, and analyzing typically rely on one or two security capabilities, each with a well-defined *Object*. Conversely, complex security intents like guaranteeing, tracing, and countering require the coordination of multiple security capabilities to be effectively carried out.

Our security capability model and security intent mapping graph are designed based on existing security knowledge and AISFs. However, this model and graph are scalable. By adding and modifying security capabilities and actions for security tasks, our proposed intent translation method can be adapted to a wider range of security scenarios.

3) SECURITY CAPABILITY REQUIREMENT

The user intent translation system converts user security intent into a security capability requirement, which we represent as follows:

$$SecurityCapabilityRequirement \Rightarrow (Iid, Conditions, SLR, SCs). \quad (2)$$

where *Iid*, *Conditions*, and *SLR* inherit the content associated with the user security intent, while *SCs* denotes the required security capabilities.

4) INTENT TRANSLATION PROCESS

The intent translation process consists of three steps. (i) The user intent translation system extracts the *Conditions*, *Object*, *Action*, and *SLR* from the user intent and assigns the identification *Iid* to the intent. (ii) The *Action* in the user intent is mapped to security capabilities using the security intent mapping graph. The required security capabilities are refined into various security sub-capabilities (*SCs*) based on the security capability model and *Object*. (iii) The components *Iid*, *Conditions*, *SLR*, and *SCs* combine to form the security capability requirement, which is then transmitted to the decision plane.

C. DECISION PLANE

1) KNOWLEDGE BASE

The knowledge base comprises SCI, AISFI, and SNI.

SCI is defined as follows:

$$SCI \triangleq \{SC_n, SC_f, SC_r\}. \tag{3}$$

where SC_n represents the name of the security capability, SC_f denotes AISFs associated with the security capability, and SC_r captures the relationships between the security capability and other security capabilities. SC_r includes subordination, dependency, and combination relationships. Subordination refers to the relationship between sub-capabilities and superior security capabilities in the security capability model. For example, attack detection belongs to passive defense. Dependency signifies that a certain security capability is a prerequisite for realizing another security capability, such as attack source tracing relying on attack detection. The combination relationship indicates that a certain security capability is realized through a combination of multiple security capabilities, such as DDoS attack detection, Web attack detection, and other capabilities that jointly implement attack detection.

AISFs are provided by developers who register with the AISF manager and submit their AISF models. The AISF manager evaluates the workload and quality of each AISF model. Workload is measured as the computation required by the model to perform inference on a batch of samples, assuming the batch size is α. Quality is assessed through performance indicators on the unified test set. For example, classification models can be evaluated using accuracy, precision, and recall, while regression models can be evaluated using mean absolute error and mean squared error. If the AISF model quality does not meet the security level, it is returned by the AISF manager. If the quality meets the security level, the AISF manager accepts the model and stores relevant information in the knowledge base. AISFI is defined as follows:

$$AISFI \triangleq \{AISF_{id}, AISF_a, AISF_p, AISF_s, AISF_w\}. \tag{4}$$

where $AISF_{id}$ represents the AISF identification assigned by the AISF manager, $AISF_a$ denotes the attribute, $AISF_p$ indicates the prerequisite (a set of other AISFs that need to be pre-deployed before deploying the AISF), $AISF_s$ represents the security level set of all AISF models, and $AISF_w$ denotes the workload set of all AISF models. The attribute $AISF_a$ can be categorized as packet-based or flow-based, depending on the feature extraction method. Packet-based AISF extracts feature vectors from an individual data packet, where each packet corresponds to one sample. Flow-based AISF aggregates packets with the same five-tuple (source/destination IP, source/destination port, and protocol) within a specific period into flows and extracts feature vectors from the aggregated flows. Multiple packets aggregated into one flow correspond to one sample.

The same AISF can have multiple models with different security levels and workloads. The security level set $AISF_s = \{(Model_{id}, s)\}$ and the workload set $AISF_w = \{(Model_{id}, w_s)\}$, where $Model_{id}$ represents the model identification, s represents the security level of the model, and w_s represents the workload of the model. Security levels are classified by the AISF manager based on the quality of the models. For example, a classification model may be classified as level 1, 2, or 3 based on its accuracy falling within specific intervals, such as (80,90], (90,95], or (95,100]. The *SLR* of the user security intent indicates that the security levels of all AISF models implementing the security intent meet the required values.

SNI is defined as follows:

$$SNI \triangleq \{SN_t, SN_r, SN_s\}. \tag{5}$$

where SN_t represents the substrate network topology, including nodes (ingress/egress nodes and function nodes) and links between nodes. SN_r indicates the substrate network resources, encompassing available computing resources of function nodes, and available bandwidth of links. SN_s represents the substrate network status, including the devices connected to the network and statistics of the communication process of each device, such as average packet length, peak traffic rate, and average number of packets corresponding to each flow. The SNI is periodically updated through network data collected by the monitoring interface.

2) AISFC POLICY

The security policy generator leverages the SCI and AISFI from the knowledge base to translate the security capability requirement into an AISFC policy. The AISFC policy is expressed as:

$$AISFCPolicy \Rightarrow (Iid, Conditions, Chain). \tag{6}$$

where I_{id} and *Conditions* inherit the relevant content from the security capability requirement. *Chain* represents an AISFC that fulfills the security capability requirement. Each AISF in *Chain* includes the AISF identification, attribute, model identification, model workload, and sequence information. The security policy generator determines the sequence information of the AISF based on the AISF's prerequisite $AISF_p$.

3) AISFC EMBEDDING POLICY

The security policy generator utilizes the AISFC embedding algorithm to translate the AISFC policy into an AISFC embedding policy, leveraging the SNI from the knowledge base. The details of the AISFC embedding algorithm can be found in Section IV. The AISFC embedding policy is expressed as:

$$AISFCEmbeddingPolicy \Rightarrow (Iid, Conditions, NEmbedding, LEmbedding). \tag{7}$$

where *Iid* and the time information in *Conditions* inherit the content of the corresponding AISFC policy. The network device range in *Conditions* is mapped to the IP

address of the corresponding device based on the SNI. *NEmbedding* represents the node embedding information, while *LEmbedding* represents the link embedding information. For the ingress/egress node, the embedding information is the IP address of the corresponding substrate network ingress/egress node. For the AISF, the embedding information includes the AISF identification, model identification, sequence information, and the IP address of the corresponding substrate network function node. For virtual links, the embedding information consists of the set of IP addresses of the substrate nodes through which the link passes.

4) POLICY GENERATION PROCESS

The policy generation process consists of six steps. (i) The security policy generator extracts the *Iid*, *Conditions*, *SLR*, and *SCs* from the security capability requirement. (ii) The security policy generator queries the SCI from the knowledge base to retrieve other security capabilities that are dependent on or combined with the required security capabilities, along with the corresponding AISF identifications. It also queries the AISFI to obtain model identifications, model workloads, and prerequisites that comply with the *SLR*. It determines the sequence information between AISFs based on $AISF_p$. These details collectively form the *Chain*. (iii) *Iid*, *Conditions*, and *Chain* together constitute the AISFC policy. (iv) The security policy generator queries the SNI to convert the network device range specified in the AISFC policy's *Conditions* into the corresponding IP addresses of the devices. (v) The security policy generator queries the SNI, invokes the AISFC embedding algorithm, and generates *NEmbedding* and *LEmbedding*. (vi) *Iid*, *Conditions*, *NEmbedding*, and *LEmbedding* combine to form the AISFC embedding policy, which is then delivered to the control plane.

Based on the AISFC embedding policy, the NFV manager within the control plane generates and deploys AISF instances. The controller generates and applies flow tables to implement the AISFC on the data plane.

IV. AISFC EMBEDDING

In this section, we address the AISFC embedding problem, which involves generating the AISFC embedding policy. To tackle this problem, we formulate it as an ILP task and propose a heuristic solution.

A. SYSTEM MODEL

The AISFC policy and SNI can be described using the following mathematical language.

1) AISFC POLICY

The *Chain* in the policy can be represented as an ordered set of virtual nodes, denoted as $\mathcal{F} = \{0, 1, 2, \ldots, F, F+1\}$. In this set, 0 and $F+1$ represent the ingress and egress nodes, respectively, while the remaining nodes represent AISFs. The virtual link between two adjacent nodes, u and v, is denoted as uv. Each AISF processes a batch of samples, with the batch size α. The set of optional security levels for all AISF models is denoted as $\mathcal{S}$ ($|\mathcal{S}| = S$). A model of AISF u with security level s has a workload w_s^u. An AISFC represents a user's security intent ($SLR = s^{th}$), where the security level of each AISF model is required to be s^{th}.

2) SNI

The SN_t can be represented as a directed graph, $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N} = \{1, 2, \ldots, N\}$ is the set of substrate nodes and $\mathcal{L}$ ($|\mathcal{L}| = L$) is the set of substrate links. If there is a link connecting substrate nodes i and j, it is denoted as link ij. The substrate nodes are divided into ingress/egress nodes, denoted as $\mathcal{N}_t$ ($|\mathcal{N}_t| = N_t$), and function nodes, denoted as $\mathcal{N}_f$ ($|\mathcal{N}_f| = N_f$). Ingress/egress nodes are responsible solely for traffic classification and forwarding and cannot deploy AISFs. SN_r includes the available computing resources C_i for an AISF on the function node i, as well as the available bandwidth resources B_{ij} for the substrate link ij. We assume that the available computing resources on the function nodes of the substrate network follow a truncated Gaussian distribution with upper and lower limits, denoted as $C \sim \mathbb{N}(\mu, \sigma^2, C_{\min}, C_{\max})$. Where μ represents the mean value, σ represents the standard deviation, and $C_{\min}$ and $C_{\max}$ represent the minimum and maximum computing resources available to function nodes, respectively. A higher standard deviation σ indicates a greater disparity in computing resources across function nodes. SN_s includes parameters such as the average length of data packets P, the peak traffic rate, and the average number of data packets per flow. The security policy generator considers the total traffic peak rate of all devices within *Conditions* as the bandwidth requirement b of the AISFC.

Next, we model the response delay of AISFs within the AISFC. When AISFs are deployed as VNFs, they typically utilize port mirroring to duplicate traffic. The inference process of AISFs for samples does not impact the forwarding of corresponding packets or flows on subsequent nodes. Given these characteristics, the response delay of an AISF in an AISFC is the duration between the packets' departure from the ingress node and the AISF's completion of generating inference results for all samples in the associated batch. The response delay encompasses both computing delay and communication delay.

The computing delay of AISF u can be defined as

$$d_u^c = \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{N}_f} x_i^u \frac{m_s^u w_s^u}{C_i}. \tag{8}$$

where the decision variable $x_i^u \in \{0, 1\}$ denotes whether AISF u is mapped to the substrate node i, and $m_s^u \in \{0, 1\}$ denotes whether AISF u selects a model with the security level s.

Similarly, the communication delay of AISF u can be described as

$$d_u^l = \sum_{v=1}^{u} \sum_{ij \in \mathcal{L}} y_{ij}^{(v-1)v} \frac{\alpha \beta P}{B_{ij}}. \tag{9}$$

where the decision variable $y_{ij}^{uv} \in \{0, 1\}$ indicates whether the substrate path, mapped by the virtual link uv, includes the substrate link ij. The variable β denotes the number of data packets associated with a sample. When the AISF attribute is packet-based, β equals 1. If the AISF attribute is flow-based, β equals the average number of packets per flow.

Consequently, the response delay of AISF u can be calculated as

$$d_u = d_u^c + d_u^l. \tag{10}$$

B. PROBLEM FORMULATION

To ensure timely response to security incidents, the total response delay of AISFs must be as short as possible. Therefore, our optimization objective is to minimize the total response delay of AISFs within the AISFC. We model this as an ILP problem. The following constraints are considered:

$$\sum_{i \in \mathcal{N}_f} x_i^u = 1, \forall u \in \mathcal{F} - \{0, F+1\}, \tag{11}$$

$$\sum_{i \in \mathcal{N}_t} x_i^u = 1, \forall u \in \{0, F+1\}, \tag{12}$$

$$\sum_{u \in \mathcal{F}} x_i^u \leq 1, \forall i \in \mathcal{N}, \tag{13}$$

$$\sum_{ij \in \mathcal{L}} y_{ij}^{(u-1)u} - \sum_{ji \in \mathcal{L}} y_{ji}^{(u-1)u} = x_i^{u-1} - x_i^u, \quad \forall i \in \mathcal{N}, u \in [1, F+1], \tag{14}$$

$$\sum_{n \in \mathcal{N}} m_u^s = 1, \forall u \in \mathcal{F} - \{0, F+1\}, \tag{15}$$

$$m_u^s s = s^{th}, \forall u \in \mathcal{F} - \{0, F+1\}, \tag{16}$$

$$\sum_{u=1}^{F+1} y_{ij}^{(u-1)u} b \leq B_{ij}, \forall ij \in \mathcal{L}. \tag{17}$$

Constraint (11) and (12) ensure the mapping of an AISF or ingress/egress to a substrate network function node or ingress/egress node, respectively. To prevent single points of failure, constraint (13) restricts each substrate node to carry only one virtual node within the AISFC. The flow continuity constraint, denoted by constraint (14), guarantees that the virtual link is mapped to a continuous substrate path. Constraint (15) and (16) enforce the condition that the security level of each selected model for an AISF matches the user's security level requirement. Lastly, constraint (17) represents the bandwidth constraint.

To summarize, the model $\text{ILP}_{\text{AISFC}}^{\text{C}}$ of the AISFC embedding problem is

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{u \in \mathcal{F} - \{0, F+1\}} d_u^c + d_u^l$$
$$\boldsymbol{s.t.}\ (10) - (16). \tag{18}$$

Existing research on the SFC embedding problem often assumes that any VNF deployment meeting the computing resource requirement is acceptable in terms of computing delay. Building upon this assumption, we introduce the baseline model $\text{ILP}_{\text{AISFC}}^{\text{C-assu}}$, which incorporates computing resource constraints as follows:

$$\sum_{\forall u \in \mathcal{F} - \{0, F+1\}} x_i^u c \leq C_i, i \in \mathcal{N}. \tag{19}$$

where c represents the computing resource requirement.

With constraint (19), the computing delay of an AISF deployed on any node meeting the computing resource requirement c is deemed acceptable. Consequently, the optimization objective of minimizing the total response delay is equivalent to minimizing the total communication delay within $\text{ILP}_{\text{AISFC}}^{\text{C-assu}}$, as shown below.

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{u \in \mathcal{F} - \{0, F+1\}} d_u^l$$
$$\boldsymbol{s.t.}\ (11) - (17), (19). \tag{20}$$

C. HEURISTIC SOLUTION

The above ILP is known to have NP-Hard properties. To address the AISFC embedding problem more efficiently, we decompose it into two sub-problems: AISF placement and routing. This decomposition allows us to devise a heuristic algorithm called AISF Placement and Routing with Ranking and Fallback (APR^2F).

The APR^2F algorithm begins by solving the AISF placement problem. The objective is to minimize the total response delay of AISFs. Given that AISFs with larger workloads require more network communication and computing resources, we introduce a ranking strategy. AISFs and function nodes are ranked based on their workload and available resources, respectively. The ranking of function nodes considers both communication and computing resources.

The algorithm scores function nodes using a combination of their computing resource score and centrality score, as shown below.

$$\gamma_i = \nu_i + \rho_i. \tag{21}$$

where ν_i indicates the node computing resource score and ρ_i indicates the node centrality score.

The computing resource score ν_i represents the ratio of available computing resources of a function node to the maximum available computing resources among all nodes. It ranges between 0 and 1. The centrality score ρ_i reflects the node's importance in the network and is calculated using normalized Closeness Centrality (CC) [43]. After obtaining scores for function nodes, the algorithm sorts them in descending order. AISFs with heavy workloads are then placed on function nodes with high scores to form the initial AISF embedding scheme.

Next, the algorithm addresses the routing problem between AISFs and the ingress/egress nodes. Routing between embedded AISFs is performed in the order specified in the AISFC. The algorithm uses a shortest path algorithm weighted by inverse bandwidth to embed virtual links between AISFs. If the remaining bandwidth of a selected NFV link is insufficient, a fallback mechanism is employed. The

algorithm adjusts the placement position of the subsequent AISF connected to the virtual link, prioritizing the function node with the highest score, and then re-routes. If the link bandwidth still does not meet the requirement and no function nodes are available for adjustment, the embedding fails.

Regarding ingress and egress, APR^{2}F employs the shortest path algorithm weighted by inverse bandwidth to select ingress/egress nodes and links with sufficient bandwidth and shortest paths. If no ingress/egress node meets the requirements, the embedding process fails.

Algorithm 1 presents the pseudocode for the APR^{2}F algorithm. In Line 2, the node_ranking() function is called to calculate function node scores and sort them in descending order. Line 3 invokes the aisf_ranking() function to rank AISFs based on their workload, from large to small. Line 4 utilizes the get_initial_placement() function to obtain the initial embedding scheme $\mathcal{F}_E^{\text{AISF}}$ for AISFs. Lines 6-15 involve the routing() function, which establishes the virtual link embedding scheme $\mathcal{L}^{u,u+1}$ between two AISFs while ensuring bandwidth constraints are met. If $\mathcal{L}^{u,u+1} = \emptyset$ and function nodes are available for adjustment $\mathcal{N}_f^{rank} \neq \emptyset$, the algorithm calls the fallback() function to modify the AISF embedding scheme and re-route. Line 20 calls the routing() function to obtain the ingress/egress node embedding scheme $\mathcal{F}_E^{gress}$, as well as the connected virtual link embedding scheme $\mathcal{L}_E^{gress}$, that satisfy the bandwidth requirements. Finally, if the embedding process is successful, the algorithm returns the node embedding scheme, $\mathcal{F}_E$, and the link embedding scheme, $\mathcal{L}_E$. The AISFC embedding policy's security policy generator generates node embedding information (*NEmbedding*) and link embedding information (*LEmbedding*) based on $\mathcal{F}_E$ and $\mathcal{L}_E$ obtained through the APR^{2}F algorithm.

Complexity Analysis: The APR^{2}F algorithm consists of AISF placement and routing. In the AISF placement stage, function node scores are calculated and sorted. The complexity of calculating the resource score for each function node is $O(N_f)$. The centrality score of function nodes is determined using the Dijkstra algorithm, with a complexity of $O(N \log N + L)$ when employing a Fibonacci heap [35] as a priority queue. Therefore, the overall complexity of calculating centrality scores for function nodes is $O(N_f(N \log N + L))$. Sorting function nodes using the bisection method has a complexity of $O(N_f^2)$. Similarly, sorting AISFs using the bisection method has a complexity of $O(F^2)$. Finally, obtaining the initial placement of AISFs has a complexity of $O(F)$.In the routing stage, Dijkstra's algorithm is again utilized. In the best-case scenario, where the fallback mechanism is not required, the complexity is $O((F-1)(N \log N + L))$. In the worst-case scenario, where the fallback mechanism needs to traverse all function nodes, the complexity is $O(N_f(N \log N + L))$. Additionally, routing between AISFs and the ingress/egress nodes occurs with a $O(2(N \log N + L))$ complexity. In summary, the minimum complexity of the APR^{2}F algorithm is $O(F + F^2 + N_f + N_f^2 + (N_f + F + 1)(N \log N + L))$, while the maximum complexity is $O(F+F^2+2N_f+N_f^2+(2N_f+2)(N \log N+L))$, which can be simplified to $O((N_f + F)(N \log N + L))$ and $O(2N_f(N \log N + L))$ respectively.

Algorithm 1 APR^{2}F

Input: AISFC Policy and SCI;
Output: $\mathcal{F}_E$ and $\mathcal{L}_E$;
1: $\mathcal{F}_E \leftarrow \emptyset$, $\mathcal{L}_E \leftarrow \emptyset$;
2: $\mathcal{N}_f^{rank}$ = node_ranking($\mathcal{N}_f, \mathcal{G}, C_i, B_{ij}$);
3: $\mathcal{F}_{\text{AISF}}^{rank}$ = aisf_ranking($\mathcal{F}, s^{th}, Q$);
4: $\mathcal{F}_E^{\text{AISF}} \leftarrow$ get_initial_placement($\mathcal{N}_f^{rank}, \mathcal{F}_{\text{AISF}}^{rank}$);
5: $\mathcal{N}_f^{rank} \leftarrow \mathcal{N}_f^{rank} - \mathcal{F}_E^{AISF}$, $u \leftarrow 1$;
6: **while** $u < F$ **do**
7: $\quad\mathcal{L}^{u,u+1} \leftarrow$ routing($\mathcal{G}, \mathcal{F}_E^{\text{AISF}}, \mathcal{L}_E, b$);
8: $\quad$**if** $\mathcal{L}^{u,u+1} = \emptyset$ **then**
9: $\quad\quad$**if** $\mathcal{N}_f^{rank} \neq \emptyset$ **then**
10: $\quad\quad\quad\mathcal{F}_E^{\text{AISF}} \leftarrow$ fallback($\mathcal{N}_f^{rank}, u+1$);
11: $\quad\quad\quad\mathcal{N}_f^{rank} \leftarrow \mathcal{N}_f^{rank} - \mathcal{F}_E^{AISF}$;
12: $\quad\quad$**else**
13: $\quad\quad\quad$**return** Embedding Failed;
14: $\quad\quad$**end if**
15: $\quad$**else**
16: $\quad\quad\mathcal{L}_E \leftarrow \mathcal{L}_E \cup \mathcal{L}^{u,u+1}$, $u \leftarrow u+1$;
17: $\quad$**end if**
18: **end while**
19: $\mathcal{F}_E \leftarrow \mathcal{F}_E \cup \mathcal{F}_E^{\text{AISF}}$;
20: $\mathcal{F}_E^{gress}, \mathcal{L}_E^{gress} \leftarrow$ routing($\mathcal{G}, \mathcal{F}_E, \mathcal{L}_E, b$);
21: **if** $\mathcal{F}_E^{gress} = \emptyset or \mathcal{L}_E^{gress} = \emptyset$ **then**
22: $\quad$**return** Embedding Failed;
23: **else**
24: $\quad\mathcal{F}_E \leftarrow \mathcal{F}_E \cup \mathcal{F}_E^{gress}, \mathcal{L}_E \leftarrow \mathcal{L}_E \cup \mathcal{L}_E^{gress}$;
25: **end if**
26: **return** $\mathcal{F}_E, \mathcal{L}_E$.

V. EXPERIMENTS

The proposed mechanism, models, and algorithms are evaluated through a prototype system and numerical simulations in this section.

A. SYSTEM AND PARAMETER SETTINGS

We implement the prototype system on DELL PowerEdge R720 servers, which have an Intel Xeon E5-2609 CPU (1.70 GHz), 32GB memory, and 1TB hard disk storage. The user plane and decision plane are implemented in Python 3.7, while the NFV manager of the control plane is based on Docker Swarm, and the controller uses OpenDaylight. For the data plane, we create virtual machines with the Ubuntu 15.04 operating system on the VMware VSphere2 virtualization platform. These virtual machines serve as the ingress/egress and function nodes, running OVS-2.6.1 with Yiyang's NSH patch [44]. The AISF models are implemented using TensorFlow 2.2, and AISF instances are deployed using Docker technology.

TABLE 1. The hyperparameters of AISF models.

AISF	Class	Model	Algo	Layers
AISF-1	Normal, ACK flood, UDP flood, others.	M-1-1	DNN	1 input layer, 1 fully connected layer, and 1 output layer.
		M-1-2	DNN	1 input layer, 2 fully connected layers, and 1 output layer.
		M-1-3	CNN	1 input layer, 2 convolutional and pooling layers, 1 fully connected layer, 1 output layer.
AISF-2	Normal, CC, SlowRead others.	M-2-1	DNN	1 input layer, 1 fully connected layer, and 1 output layer.
		M-2-2	CNN	1 input layer, 2 convolutional and pooling layers, 1 fully connected layer, 1 output layer.
		M-2-3	CNN	1 input layer, 4 convolutional and pooling layers, 2 fully connected layers, 1 output layer.
AISF-3	Normal, HTTP-Get, others.	M-3-1	DNN	1 input layer, 2 fully connected layer, and 1 output layer.
		M-3-2	CNN	1 input layer, 3 convolutional and pooling layers, 2 fully connected layer, 1 output layer.
		M-2-3	CNN	1 input layer, 4 convolutional and pooling layers, 2 fully connected layers, 1 output layer.
Activation function		**Optimizer, epochs, learning rate**		
ReLU, Softmax(output layer).		Adam, 40, 0.001.		

TABLE 2. Security capability information and AISF information.

SCI					
SC_n		SC_f		SC_r	
DDoS attack detection		AISF-1, AISF-2, AISF-3		DDoS attack detection-attack detection-passive defense	
AISFI					
$AISF_{id}$	$AISF_a$	$AISF_p$	$Model_{id}$	$AISF_s$	$AISF_w(10^6)$
AISF-1	flow-based	/	Model-1-1	1	81
			Model-1-2	2	87
			Model-1-3	3	126
AISF-2	flow-based	AISF-1	Model-2-1	1	89
			Model-2-2	2	125
			Model-2-3	3	158
AISF-3	flow-based	AISF-1, AISF-2	Model-3-1	1	86
			Model-3-2	2	145
			Model-3-3	3	238

The User Security Intent, SCI, AISFI and SNI settings in the prototype system are as follows.

1) USER SECURITY INTENT

In the prototype system, the User Security Intent is set to detect DDoS attacks in communication traffic with the edge server, based on our team's previous research [12]. We evaluate the translation and implementation results of the user security intent, as well as the performance of models and algorithms for *SLR* values s^{th} of 1, 2, and 3.

2) SCI AND AISFI

The security capability involved in the experiment is DDoS attack detection, which falls under attack detection of passive defense. Three AISFs (AISF-1, AISF-2, and AISF-3) are designed based on DNN [11] and CNN [12] models (the model hyperparameters of AISF models are shown in the TABLE 1), each with different security levels. The models are trained and verified on a DDoS attack dataset collected by our team [12]. The dataset is divided into a training set and two test sets (test set 1 and test set 2). Test set 1 serves as a unified test set in AISF Manager to evaluate the quality and workload of AISF models, while test set 2 is used to evaluate the performance of deployed AISF models. The AISFs integrate CICFlowMeter [45] as feature extraction tools, making their attributes flow-based. AISF-1 is a prerequisite for AISF-2, and both AISF-1 and AISF-2 are prerequisites for AISF-3.

The AISF models discussed here are classification models, and their quality can be assessed by the AISF manager using accuracy metrics. The AISF manager determines the security level of a model based on its performance on test set 1. Models with accuracy within the range (80, 90] are assigned security level 1, accuracy within (90, 95] are assigned security level 2, and accuracy within (95, 100] are assigned security level 3. The workload of the AISF models is measured by the average inference time per batch of samples, multiplied by the available computing resources of the running node.

In summary, the SCI and AISFI settings are shown in TABLE 2.

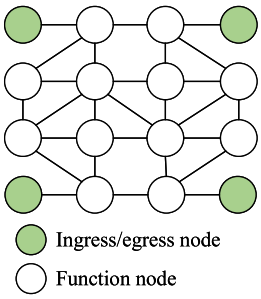


FIGURE 5. The substrate network topology.

3) SNI

Regarding the substrate network topology (shown in FIGURE 5), there are 12 function nodes (white nodes) and 4 ingress/egress nodes (green nodes). CPU Limit [46] is used to set maximum CPU usage for function node processes, reflecting different available computing resources per node (available computing resources = CPU cycles per second × CPU usage). The CPU usage distribution of nodes follows a truncated Gaussian distribution, ranging from a minimum value $C_{\min}$ of 20% to a maximum value $C_{\max}$ of 100%. The mean CPU usage μ is 70%, and the standard deviation is σ. Substrate link bandwidth is set using Linux Traffic Control, with randomly selected values from [100, 200, 300, 400] Mbit/s. The test set 2 was used to simulate communication with the edge server, with measurements indicating an average packet length P of 5442 bits, an average packet number per flow β of 1.47, and a total traffic peak rate

of 100 Mbit/s. Therefore, the bandwidth requirement b for AISFC is set to 100 Mbit/s.

The comparison and numerical simulation settings are as follows.

4) COMPARISION SETTINGS

(i) In the model comparison, we utilize Gurobi to solve ILP^{C-assu}_{AISFC} and ILP^{C}_{AISFC}, analyzing the performance of these two ILP formulations under different *SLR*s and computing resource distributions. (ii) In the algorithm comparison, we employ the APR^{2}F algorithm along with two other solutions, namely Gurobi and the multi-layer graph-based greedy algorithm (Greedy), to address the problem. We evaluate the performance of these three solutions considering various *SLR*s, computing resource distributions, and network scales (through numerical simulation). Gurobi [47] employs an exact solution framework, providing optimal solutions. The Greedy algorithm, proposed in [32], selects the path and node with the minimum cost to connect two adjacent layers on a multi-layer graph. We modified the algorithm to suit the AISFC embedding problem, selecting the path and node that satisfy the constraints while minimizing the response delay. Greedy serves as the baseline algorithm.

5) NUMERICAL SIMULATION SETTINGS

The Numerical simulation settings are designed to evaluate the time complexity of the algorithms. Execution times of APR^{2}F, Greedy, and Gurobi are tested under different substrate network scales. In the simulation experiments, APR^{2}F and Greedy are implemented in C++ to match the underlying C++ programming language used by Gurobi. The substrate network scale ranges from a minimum of 100 nodes to a maximum of 1600 nodes, with a step size of 100. The ratio of function nodes to ingress/egress nodes was set at 3:1. Link bandwidth is randomly selected from [0, 100, 200, 300, 400], where 0 indicates no link between the two nodes. The test host configuration consists of an Intel Core i7-8700 CPU (6 cores and 3.20GHz) and 16GB memory.

B. RESULTS AND DISCUSSION

1) USER SECURITY INTENT TRANSLATION AND IMPLEMENTATION RESULTS

FIGURE 6 illustrates the hierarchical translation process and results of the user security intent when the *SLR* is set to 1. For *SLR* 2 and 3, the translation process follows a similar pattern as depicted in FIGURE 6.

The user intent translation process in the user plane involves the following steps: (i) The user intent translation system extracts *Conditions* $\Rightarrow$ (Source = "any", Destination = "Edge server"), *Object* $\Rightarrow$ (Object = "DDoS Attack"), *Action* $\Rightarrow$ (Action = "Detect"), and *SLR* $\Rightarrow$ (SLR = 1) from the user security intent and assigns an identification *Iid* $\Rightarrow$ (Iid = "IntentID") to the intent. (ii) Using the security intent mapping graph, the Action="Detect" is mapped to passive defense in the security capability model. Based on the Object="DDoS Attack" and the security capability model, passive defense is further refined into DDoS detection in attack detection. (iii) *Iid*, *Conditions*, *SLR*, and *SCs* $\Rightarrow$ (SC = "DDoS detection") form the security capability requirement, which is then forwarded to the decision plane.

TABLE 3. User security intent implementation results.

User security intent	*SLR*	AISFC	Accuracy (%)	
Detect DDoS attacks in communication traffic with the edge server.	1	(Ingress)-(M-1-1)-(M-2-1)-(M-3-1)-(Egress)	M-1-1	84.14
			M-2-1	83.01
			M-3-1	85.16
	2	(Ingress)-(M-1-2)-(M-2-2)-(M-3-2)-(Egress)	M-1-2	90.72
			M-2-2	93.06
			M-3-2	91.79
	3	(Ingress)-(M-1-3)-(M-2-3)-(M-3-3)-(Egress)	M-1-3	96.46
			M-2-3	95.07
			M-3-3	95.55

The policy generation process in the decision plane consists of the following steps: (i) The security policy generator extracts *Iid*, *Conditions*, *SLR*, and *SCs* from the security capability requirement. (ii) The security policy generator queries the knowledge base for the SCI and the AISFI to obtain the AISFC *Chain* $\Rightarrow$ (Ingress ="ingress", (AISFid ="AISF-1", AISFa="flow-based", Modelid= "M-1-1", Workload = 81, Order=1),..., Egress="egress") that meets the requirement. (iii) *Iid*, *Conditions*, and *Chain* form the AISFC policy. (iv) The security policy generator queries the SNI in the knowledge base and converts *Conditions* into *Conditions* $\Rightarrow$ (Source = "any", Destination = "27.0.4.3"). (v) The security policy generator queries the SNI. It employs the AISFC embedding algorithm to generate *NEmbedding* $\Rightarrow$ (Ingress = "27.0.1.2", (AISFid = "AISF-1", Modelid = "M-1-1", Order = 1, IPv4 = "27.0.1.6"),...., Egress = "27.0.4.2") and *LEmbedding* $\Rightarrow$ ((IPv4 = "27.0.1.2", IPv4 = "27.0.1.6", Order = 1),...). (vi) *Iid*, *Conditions*, *NEmbedding*, and *LEmbedding* form the AISFC embedding policy, which is then sent to the control plane.

Note that FIGURE 6 displays a partial representation of the user security intent, security capability requirement, AISFC policy, and AISFC embedding policy. For complete information, see https://github.com/wwl0220/AISSADM.

Based on the AISFC embedding policy, the NFV manager in the control plane generates and deploys AISF instances, while the controller generates and delivers flow tables. Subsequently, we evaluate the implementation of the user security intent using test set 2. The evaluation results are presented in TABLE 3.

From the evaluation results in TABLE 3, we observe that the accuracy rates of the deployed AISF model instances (M-1-1, M-2-1, and M-3-1) are all above 80%. Moreover, the deployed AISF model instances (M-1-2, M-2-2, and M-3-2) achieve accuracy rates exceeding 90%. Additionally, the deployed AISF model instances (M-1-3, M-2-3, and M-3-3) exhibit accuracy rates surpassing 95%. Consequently,

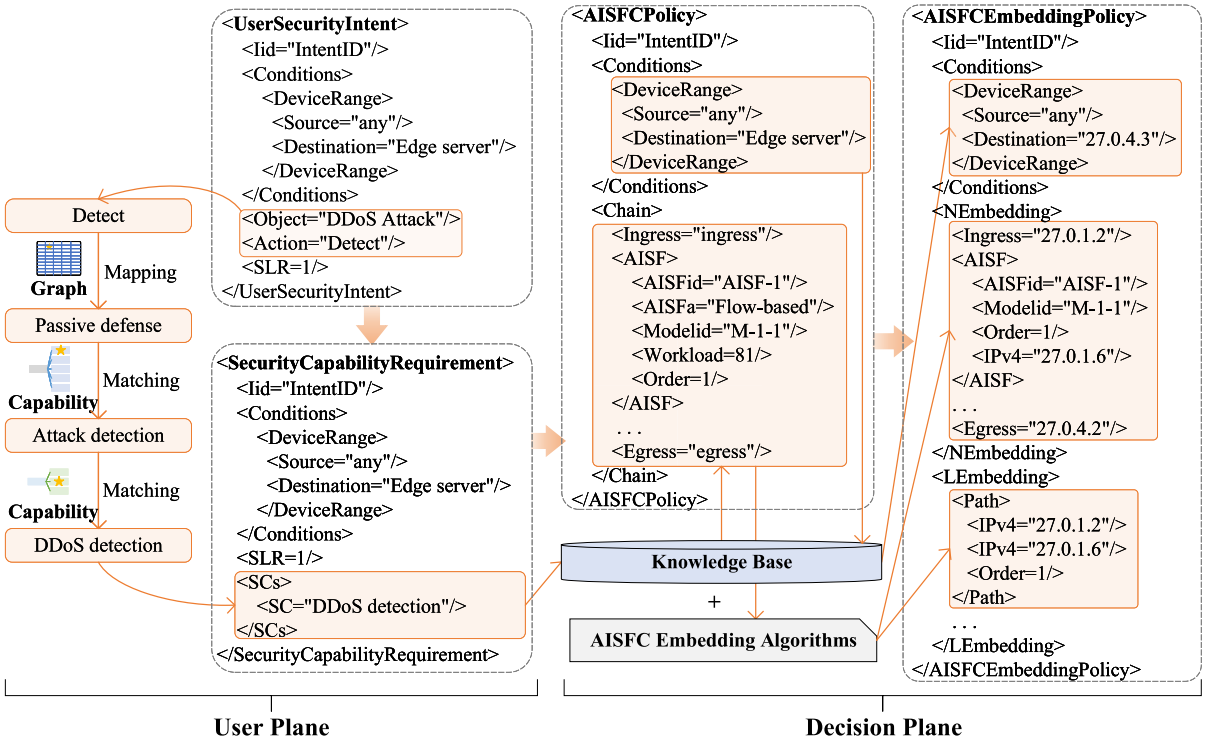
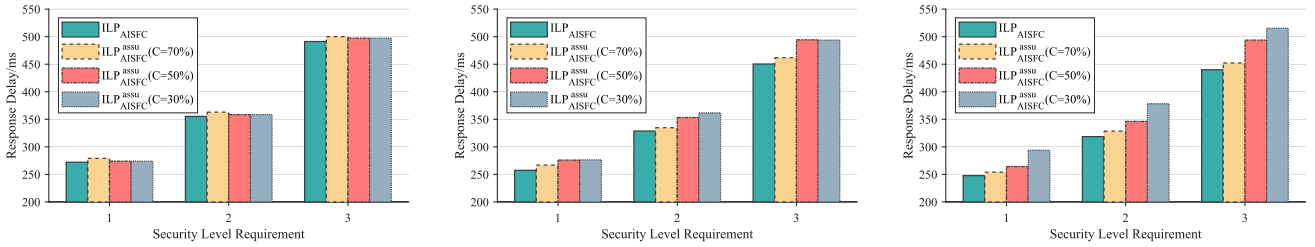


FIGURE 6. The hierarchical translation process and results of the user security intent.



(a) σ=2 (b) σ=10 (c) σ=20

FIGURE 7. The average total response delay calculated by Gurobi while solving $\text{ILP}^{\text{C}}_{\text{AISFC}}$ and $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ for security level requirements 1, 2, and 3. The computing resource standard deviations considered are (a) 2, (b) 10, and (c) 20.

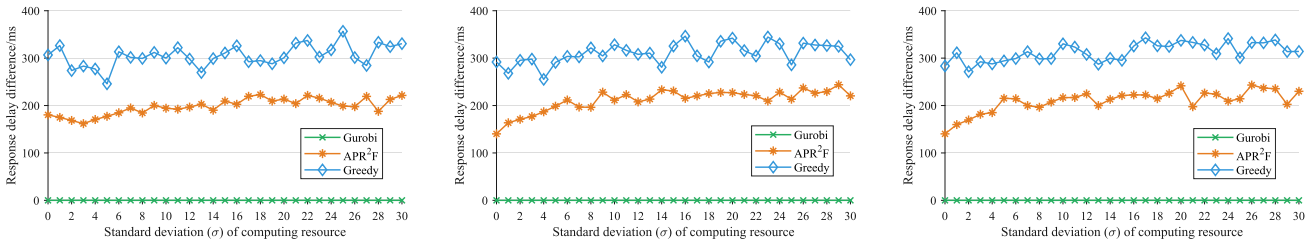
all deployed AISF model instances meet the user's security level requirement.

2) MODEL EVALUATION RESULTS

In FIGURE 7, we present the average total response delay calculated by Gurobi while solving $\text{ILP}^{\text{C}}_{\text{AISFC}}$ and $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ for security level requirements 1, 2, and 3. The computing resource standard deviations σ considered are 2, 10, and 20, respectively. For $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$, the computing resource requirements c are set to 30%, 50%, and 70% (CPU usage), respectively.

Analyzing the results in FIGURE 7, we observe the following conclusions:

- Under the same conditions (computing resource distribution and security level requirements), $\text{ILP}^{\text{C}}_{\text{AISFC}}$ achieves smaller total response delays compared to $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$. When σ is 2, $\text{ILP}^{\text{C}}_{\text{AISFC}}$ has a slight advantage over $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$. As σ increases to 10, the advantages of $\text{ILP}^{\text{C}}_{\text{AISFC}}$ over $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ gradually become more pronounced. With σ at 20 and a security level requirement of 3, $\text{ILP}^{\text{C}}_{\text{AISFC}}$ achieves response delays that are 12.33ms, 53.88ms, and 75.41ms shorter than $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ (C=70%), $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ (C=50%), and $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ (C=30%), respectively. This is because $\text{ILP}^{\text{C}}_{\text{AISFC}}$ considers the impact of both computing resources and communication resources in the entire network, while $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ only focuses on the communication resources in the network subgraph that meets the computing resource requirements. When σ is small, the difference in available computing resources between nodes is minimal, and communication resources have a dominant impact on response delay. Consequently, the disparity between $\text{ILP}^{\text{C-assu}}_{\text{AISFC}}$ and $\text{ILP}^{\text{C}}_{\text{AISFC}}$ is reduced. However, as σ increases, the difference in available computing



(a) *SLR*=1 (b) *SLR*=2 (c) *SLR*=3

FIGURE 8. The response delay differences between the heuristic algorithms (APR^{2}F and Greedy) and Gurobi for solving ILP$^{C}_{AISFC}$ vary with the standard deviation of computing resources under security level requirements of (a) 1, (b) 2, and (c) 3.

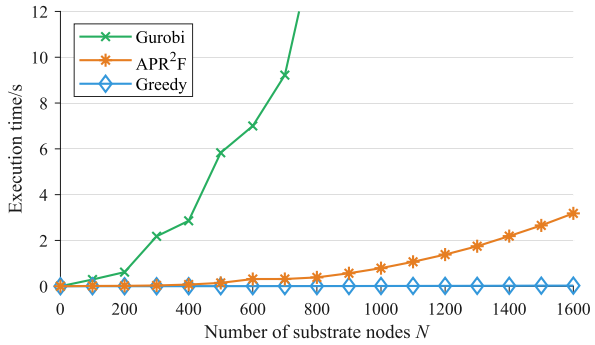


FIGURE 9. The execution time of Gurobi, APR^{2}F, and Greedy algorithm vary with the substrate network size.

resources between nodes becomes more significant, and the influence of computing resources on response delay becomes crucial. Hence, ILP$^{C}_{AISFC}$, which considers the relationship between available computing resources and delay, holds a greater advantage over ILP$^{C-assu}_{AISFC}$.

• The difference in response delays obtained by ILP$^{C}_{AISFC}$ and ILP$^{C-assu}_{AISFC}$ becomes greater as the security level requirement increases, especially when there is a significant difference in computing resources between nodes. For instance, in FIGURE 7(c), when the security level requirement is 1, ILP$^{C}_{AISFC}$ achieves an average response delay that is 6.22ms smaller than ILP$^{C-assu}_{AISFC}$ (C=70%). When the security level requirement is 2, ILP$^{C}_{AISFC}$ achieves an average response delay that is 9.99ms smaller than ILP$^{C-assu}_{AISFC}$ (C=70%). When the security level requirement is 3, ILP$^{C}_{AISFC}$ achieves an average response delay that is 12.33ms smaller than ILP$^{C-assu}_{AISFC}$ (C=70%). This is because higher security levels increase the workload of the AISF model, making it more sensitive to the available computing resources.

In summary, ILP$^{C}_{AISFC}$ comprehensively considers the impact of computing resources and communication resources in the entire network, resulting in smaller response delays compared to ILP$^{C-assu}_{AISFC}$. ILP$^{C}_{AISFC}$ is particularly suitable for resource allocation for AISF models with larger workloads.

3) ALGORITHM EVALUATION RESULTS

FIGURE 8 illustrates the response delay differences between the heuristic algorithms (APR^{2}F and Greedy) and Gurobi for solving ILP$^{C}_{AISFC}$. These differences vary with the standard deviation σ of computing resources and are evaluated under security level requirements of 1, 2, and 3, respectively.

FIGURE 8 indicates that the AISFC embedding scheme obtained by the APR^{2}F algorithm is closer to the optimal solution compared to the Greedy algorithm. Across different security levels and computing resource scenarios, the response delay difference between the APR^{2}F algorithm and Gurobi remains within 250ms, which is approximately 100ms (28.57%) shorter on average than the Greedy algorithm. This improvement can be attributed to the ranking strategy employed by the APR^{2}F algorithm, which prioritizes the deployment of AISF models with larger workloads on function nodes possessing more available computing and communication resources. By giving preference to AISF models with higher workloads, the APR^{2}F algorithm outperforms the Greedy algorithm.

FIGURE 9 presents the execution time of Gurobi, APR^{2}F, and the Greedy algorithm varying with the substrate network size in numerical simulation experiments.

Based on FIGURE 9, it can be observed that the execution time of APR^{2}F and Greedy increases at a much slower rate with network size compared to Gurobi. Additionally, the execution time of the APR^{2}F algorithm is slightly higher than that of the Greedy algorithm. Significantly, Gurobi supports less than 300 substrate network nodes within a 2-second execution time, while the APR^{2}F algorithm supports over 1,300 nodes (approximately 4.34 times larger than Gurobi) and the Greedy algorithm supports well beyond 1,600 nodes (approximately 5.34 times larger than Gurobi) within the same time frame. This discrepancy arises because heuristic algorithms provide approximate optimal solutions to the AISFC embedding problem, resulting in lower algorithm complexity compared to exact calculations. However, the APR^{2}F algorithm exhibits a slightly longer execution time than the Greedy algorithm due to increased complexity in function node centrality calculation, ranking, and fallback processes.

In summary, the APR^{2}F algorithm achieves a solution closer to optimality than the Greedy algorithm and supports significantly larger-scale networks than the exact solution within the same execution time.

VI. CONCLUSION

This paper addresses the challenges of complex service configuration and ensuring AI security service performance during autonomous deployment. We propose an autonomous deployment mechanism for AI security services that achieves the translation of user security intent and guarantees service implementation aligned with the user's security level requirement. Our approach involves an autonomous deployment mechanism with user and decision planes integrated into the control plane. By hierarchically translating user security intent into network executable policies, the proposed mechanism converts abstract intent into concrete actions. Additionally, we tackle the AISFC embedding problem by introducing the model $\text{ILP}^{\text{C}}_{\text{AISFC}}$, optimizing response delay by considering computing resource relationships. Furthermore, we propose the APR^2F algorithm, which decomposes the embedding problem and demonstrates polynomial time complexity. Through a prototype system and numerical simulations, we validate the effectiveness of our mechanism in realizing autonomous translation, implementation, and guaranteeing the user security intent. By considering the relationship between available computing resources and delay, our approach outperforms the baseline in terms of response delay. Additionally, our algorithm achieves approximately 28.57% greater proximity to the optimal solution compared to the Greedy algorithm. Furthermore, it supports a network scale 4.34 times larger than Gurobi within a 2-second execution time.

REFERENCES

[1] G. Liu, N. Li, J. Deng, Y. Wang, J. Sun, and Y. Huang, "The SOLIDS 6G mobile network architecture: Driving forces, features, and functional topology," *Engineering*, vol. 8, pp. 42–59, Jan. 2022.

[2] *5G System; Technical Realization of Service Based Architecture*, 3rd Generation Partnership Project (3GPP), Technical Specification (TS) document 29.500, 6 2023, Version 18.2.0, Nov. 2020.

[3] *Zero-Touch Network and Service Management (ZSM); Enablers for Artificial Intelligence-Based Network and Service Automation*, ETSI Group Specification (GS) document 12 2022, ETSI GS ZSM 012 Version 1.1.1, Dec. 2022.

[4] Y. Yang et al., "6G network ai architecture for everyone-centric customized services," 2022, *arXiv:2205.09944*.

[5] J. Wu, R. Li, X. An, C. Peng, Z. Liu, J. Crowcroft, and H. Zhang, "Toward native artificial intelligence in 6G networks: System design, architectures, and paradigms," 2021, *arXiv:2103.02823*.

[6] *6G AI as a Service Requirement Research*, document IMT-2030, (6G) Promotion Group Report, Apr. 2023. [Online]. Available: https://www.imt2030.org.cn

[7] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "AI and 6G security: Opportunities and challenges," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, Jun. 2021, pp. 616–621.

[8] V.-L. Nguyen, P.-C. Lin, B.-C. Cheng, R.-H. Hwang, and Y.-D. Lin, "Security and privacy for 6G: A survey on prospective technologies and challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2384–2428, 4th Quart., 2021.

[9] J. Halpern and C. Pignataro, *Service Function Chaining (SFC) Architecture*, document IETF RFC7665, Oct. 2015.

[10] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine learning-based network vulnerability analysis of Industrial Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6822–6834, Aug. 2019.

[11] A. Bhardwaj, V. Mangat, and R. Vig, "Hyperband tuned deep neural network with well posed stacked sparse AutoEncoder for detection of DDoS attacks in cloud," *IEEE Access*, vol. 8, pp. 181916–181929, 2020.

[12] M. Li, H. Zhou, and Y. Qin, "Two-stage intelligent model for detecting malicious DDoS behavior," *Sensors*, vol. 22, no. 7, p. 2532, Mar. 2022.

[13] J. Li, L. Lyu, X. Liu, X. Zhang, and X. Lyu, "FLEAM: A federated learning empowered architecture to mitigate DDoS in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4059–4068, Jun. 2022.

[14] C. Picard and S. Pierre, "RLAuth: A risk-based authentication system using reinforcement learning," *IEEE Access*, vol. 11, pp. 61129–61143, 2023.

[15] P. Fu, J. Wu, X. Lin, and A. Shen, "ZTEI: Zero-trust and edge intelligence empowered continuous authentication for satellite networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2022, pp. 2376–2381.

[16] H. Fang, A. Qi, and X. Wang, "Fast authentication and progressive authorization in large-scale IoT: How to leverage AI for security enhancement," *IEEE Netw.*, vol. 34, no. 3, pp. 24–29, May/Jun. 2020.

[17] H. Yang, K. Zhan, M. Kadoch, Y. Liang, and M. Cheriet, "BLCS: Brain-like distributed control security in cyber physical systems," *IEEE Netw.*, vol. 34, no. 3, pp. 8–15, May 2020.

[18] M. Sinha, S. Gupta, S. S. Rout, and S. Deb, "Sniffer: A machine learning approach for DoS attack localization in NoC-based SoCs," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 2, pp. 278–291, Jun. 2021.

[19] K. Li, H. Zhou, Z. Tu, O. Liu, and H. Zhang, "AT-GCN: A DDoS attack path tracing system based on attack traceability knowledge base and GCN," *Comput. Netw.*, vol. 236, Nov. 2023, Art. no. 110036. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623004814

[20] T. Mahboob, J. W. Lim, S. T. Shah, and M. Y. Chung, "A novel deep-learning-enabled QoS management scheme for encrypted traffic in software-defined cellular networks," *IEEE Syst. J.*, vol. 16, no. 2, pp. 2844–2855, Jun. 2022.

[21] X. Yun, Y. Wang, Y. Zhang, C. Zhao, and Z. Zhao, "Encrypted TLS traffic classification on cloud platforms," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 164–177, Feb. 2023.

[22] P. Quinn and U. Elzur, *Network Service Header*, document IETF RFC8300, Feb. 2018.

[23] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, *Intent-Based Networking-Concepts and Definitions*, document IETF RFC 9315, 2021.

[24] K. Abbas, T. A. Khan, M. Afaq, and W.-C. Song, "Network slice lifecycle management for 5G mobile networks: An intent-based networking approach," *IEEE Access*, vol. 9, pp. 80128–80146, 2021.

[25] A. Chowdhary, A. Sabur, N. Vadnere, and D. Huang, "Intent-driven security policy management for software-defined systems," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 4, pp. 5208–5223, Dec. 2022.

[26] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, "Automatic intent-based secure service creation through a multilayer SDN network orchestration," *J. Opt. Commun. Netw.*, vol. 10, no. 4, pp. 289–297, Apr. 2018.

[27] J. M. B. Murcia, J. F. P. Zarca, A. M. Zarca, and A. Skármeta, "By-default security orchestration on distributed edge/cloud computing framework," in *Proc. IEEE 9th Int. Conf. Netw. Softwarization (NetSoft)*, Jun. 2023, pp. 504–509.

[28] J. Kim, E. Kim, J. Yang, J. Jeong, H. Kim, S. Hyun, H. Yang, J. Oh, Y. Kim, S. Hares, and L. Dunbar, "IBCS: Intent-based cloud services for security applications," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 45–51, 2020.

[29] D. D. A. Nguyen, F. Autrel, A. Bouabdallah, and G. Doyen, "A robust approach for the detection and prevention of conflicts in I2NSF security policies," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp. (NOMS)*, May 2023, pp. 1–7.

[30] W. Zhang, S. Zeadally, W. Li, H. Zhang, J. Hou, and V. C. M. Leung, "Edge AI as a service: Configurable model deployment and delay-energy optimization with result quality constraints," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1954–1969, Apr./Jun. 2023.

[31] P. Patel, M. I. Ali, and A. Sheth, "On using the intelligent edge for IoT analytics," *IEEE Intell. Syst.*, vol. 32, no. 5, pp. 64–69, Sep. 2017.

[32] B. Yuan and B. Ren, "Embedding the minimum cost SFC with end-to-end delay constraint," in *Proc. 5th Int. Conf. Mech., Control Comput. Eng. (ICMCCE)*, Dec. 2020, pp. 2299–2303.

[33] M. Sasabe and T. Hara, "Capacitated shortest path tour problem-based integer linear programming for service chaining and function placement in NFV networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 104–117, Mar. 2021.

[34] G. Li, H. Zhou, B. Feng, Y. Zhang, and S. Yu, "Efficient provision of service function chains in overlay networks using reinforcement learning," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 383–395, Jan. 2022.

[35] S. Horimoto and E. Oki, "Virtual network function placement model considering both service delay and availability," in *Proc. 23rd Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2022, pp. 1–6.
[36] A. Varasteh, B. Madiwalar, A. V. Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-aware and delay-constrained VNF placement and chaining," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1524–1539, Jun. 2021.
[37] S. Wang, C. Yuen, W. Ni, Y. L. Guan, and T. Lv, "Multiagent deep reinforcement learning for cost- and delay-sensitive virtual network function placement and routing," *IEEE Trans. Commun.*, vol. 70, no. 8, pp. 5208–5224, Aug. 2022.
[38] Y. She, M. Li, Y. Jin, M. Xu, J. Wang, and B. Liu, "On-demand edge inference scheduling with accuracy and deadline guarantee," in *Proc. IEEE/ACM 31st Int. Symp. Quality Service (IWQoS)*, Jun. 2023, pp. 1–10.
[39] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. S. Shen, "Deep reinforcement learning based resource management for DNN inference in IIoT," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6.
[40] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for DNN inference tasks in mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11453–11464, Jul. 2023.
[41] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Apr. 2020.
[42] R. M. Lee. (2015). *The Sliding Scale of Cyber Security*. [Online]. Available: https://www.sans.org/white-papers/36240/
[43] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 3rd ed. Berlin, Germany: Springer, 2005.
[44] Y. Yi. (2017). *OVS NSH Pathes*. [Online]. Available: https://github.com/yyang13/ovs_nsh_patches
[45] (2020). *Cicflowmeter*. [Online]. Available: https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter
[46] (2015). *Cpulimit*. [Online]. Available: https://github.com/opsengine/cpulimit
[47] Gurobi. *Gurobi Optimizer 9.5*. Accessed: 2021. [Online]. Available: https://www.gurobi.com

WEILIN WANG received the B.S. degree in telecommunications engineering from Beijing Jiaotong University (BJTU), China, in 2020. She is currently pursuing the Ph.D. degree in information and telecommunications engineering. She joined the National Engineering Research Center for Advanced Network Technologies, BJTU. Her research interests include the architecture of next-generation internet, network security, and artificial intelligence.

HUACHUN ZHOU received the B.S. degree from the Peoples Police Officer University of China, in 1986, and the M.S. degree in telecommunication automation and the Ph.D. degree in telecommunications and information systems from Beijing Jiaotong University (BJTU), in 1989 and 2008, respectively. He is currently a Professor with the National Engineering Research Center for Advanced Network Technologies, BJTU. He has authored more than 40 peer-reviewed articles and he is the holder of 17 patents. His research interests include mobility management, mobile and secure computing, routing protocols, network management, and satellite networks.

MAN LI received the B.S. degree in communication engineering from Henan University, Kaifeng, China, in 2018. She is currently pursuing the Ph.D. degree with the School of Electronic Information Engineering, Beijing Jiaotong University, Beijing, China, with a focus on cyber security, service function chain, and machine learning.

JINGFU YAN received the B.S. degree from Anhui University, in June 2018, and the M.S. degree from Beijing Jiaotong University, Beijing, China, in June 2022, where he is currently pursuing the Ph.D. degree with the National Engineering Research Center for Advanced Network Technologies. His research interests include network security, next-generation internet, and intelligent networks.