# VSA-SD: A Service Discovery Method Based on Vector Symbol Architecture for Low-Cost IoT System Development

Haiming Chen, *Member, IEEE*, Lei Wang, Wei Qin, Xinyan Zhou, and Li Cui, *Member, IEEE*

***Abstract*—In recent years, with the widening applications of the Internet of Things (IoT), more and more perception services (e.g., air quality indicator services, road traffic congestion monitoring services, etc) with different arguments (e.g., data type, source location, creator, etc) will be deployed by dedicated IT infrastructure service providers for constructing customized IoT systems with low cost by subscription. So it is an indispensable step to check whether the required perception services with specified arguments have been available for the constructing IoT through discovery method to reduce the redundancy of service deployment. However, it is a challenging problem to design efficient (i.e., achieving high accuracy and low response delay with low overhead), highly robust, and trustworthy mechanisms for discovering perception services on resource-constrained IoT devices. To solve this problem, we proposed a distributed service discovery method, named VSA-SD, based on the Vector Symbolic Architecture (VSA). This method employs hyperdimensional vectors to describe services in a distributed manner, and measures the degree of service matching by calculating the Hamming distance, thereby achieving service discovery. We implemented VSA-SD in NBUFlow, which is an IoT task construction and offloading test platform, and evaluated its performance through comprehensive experiments. Results show that VSA-SD outperforms the centralized, hybrid, and other distributed service discovery mechanisms in terms of accuracy, response delay, overhead, robustness, trustability, interoperability, and mobility.**

***Index Terms*—Internet of things (IoT), service discovery, vector symbol architecture.**

## I. Introduction

RECENTLY, with the rapid development of embedded systems, hardware miniaturization, and low-cost device manufacturing, the number of smart devices is growing rapidly. These devices differ significantly in computing, storage, and energy consumption. The ubiquitous heterogeneous devices are connected to the Internet, forming the Internet of Things (IoT) and covering a wide range of fields such as smart transportation, smart grid and smart agriculture, et al. [1], which provides a near real-time state of the physical world. According to *The mobile economy 2023* report published by the Global System for Mobile Communications Association (GSMA), the number of IoT connections will double to 5.3 billion by 2030 [2].

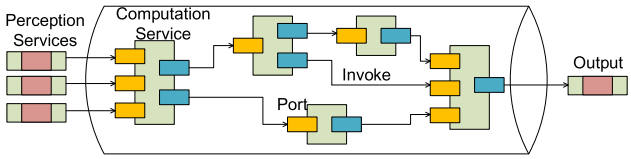


Fig. 1. Illustration of building IoT system through workflow orchestration.

Currently, in various fields, it is mainly based on cloud platforms to build IoT systems through workflow orchestration of data production functions (*perception services*) and data processing functions (*computation services*) in cloud [3], as shown in Fig. 1. Considering the increasing requirements of processing delay-sensitive tasks and improving the system performance, more and more edge devices have been deployed in IoT. So many perception services are directly provided by edge devices, while computation services can be offloaded to the edges to achieve collaborative task processing at the cloud and edge [4]. Due to the dependency of perception service on device, in other words, perception services are usually hosted in devices equipped with sensors which have appropriate data acquisition capabilities, and seldom migrate from the host devices. So perception services can be provided by dedicated IT infrastructure service providers and subscribed by multiple IoT systems. In particular, when constructing IoT systems, the required perception services should be queried using service discovery. If the services already exist and are available, it can be accessed without redeployment, so as to avoid the resource consumption caused by redundant deployment and improve device utilization [5].

From the above analysis, we can see that the service discovery mechanisms are emerging requirements for low cost development of cloud-edge collaborative IoT systems[6]. Designing service discovery mechanism refers to as defining specific policies for both service requesters and service providers. The former describes the target service or function to be used, while the latter

Manuscript received 4 June 2023; revised 14 December 2023; accepted 15 December 2023. Date of publication 19 December 2023; date of current version 8 March 2024. This work was supported in part by the Natural Science Foundation of Zhejiang Province under Grant Y24F020025, in part by Ningbo Municipal Commonweal S&T Project under Grant 2022S005, in part by the Major S&T Projects of Ningbo High-tech Zone under Grant 2022BCX05001, and in part by the National Natural Science Foundation of China under Grant 62002183. Recommended for acceptance by J. Zhai. *(Corresponding author: Haiming Chen.)*

Haiming Chen, Lei Wang, Wei Qin, and Xinyan Zhou are with the Faculty of Electrical Engineering and Computer Science, Zhejiang Key Laboratory of Mobile Network Application Technology, Ningbo University, Zhejiang 315211, China (e-mail: chenhaiming@nbu.edu.cn; 1220266172@qq.com; 2011082317@nbu.edu.cn; zhouxinyan@nbu.edu.cn).

Li Cui is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: lcui@ict.ac.cn).

Digital Object Identifier 10.1109/TCC.2023.3344512

describes the services that can be provided [7]. Although there have been many service discovery mechanisms in ubiquitous environment [8], designing such a perception service discovery mechanism, which is supposed to be semantically rich, low complexity, highly robust, and trustworthy in IoT scenarios is still a significant challenge, mainly due to the following reasons: 1) difficulty to effectively manage large number of devices, resulting in scalability problems in the large search space; 2) difficulty to perform a large amount of computation or storage due to the limited resource of devices, leading to low efficiency of service discovery; 3) proneness of leading to discovery error resulting from the unstable data transmission in the IoT environment; 4) hardness of guaranteeing the security of the requested data during the service discovery process.

Most of the currently widely used service discovery mechanisms in ubiquitous environment rely on central registries to store information about all the services, like Netflix Eureka, HashiCorp Consul, and Apache Zookeeper. Albalas [9] et al. proposed an Adaptive Fibonacci sequence-based Tuning (AFT) method for service discovery in IoT using the CoAP protocol. Kim [10] et al. proposed a DNS naming service for service discovery and remote control of IoT devices. Due to the large scale of IoT systems, all these mechanism based on registry queries have potentially growing problem in scalability and robustness of service discovery.

Additionally, Mastorakis [11] et al. proposed a distributed service discovery mechanism based on Named Data Networks (NDN). Tanganelli [12] et al. proposed an edge-centric distributed architecture to provide resource discovery and access services for IoT applications. Santos [13] et al. proposed a peer-to-peer (P2P) distributed hash table (DHT) resource discovery service, which federated multiple IoT gateways by a P2P overlay implemented through a DHT storing information about the IoT resources for global lookup. However, the DHT-based approaches have disability of fuzzy matching, and the maintenance of hash tables also brings a large overhead. Although the above distributed service discovery eliminates the disadvantages of registry-based service discovery to a certain extent, the low efficiency of service discovery due to limited resources of device nodes, proneness of discovery error resulting from unreliable data transmission, and hardness of guaranteeing the security of service discovery are not yet effectively solved in the existing distributed service discovery methods.

In this regard, this paper proposes a Vector Symbolic Architecture (VSA)-based IoT service discovery method for low-cost development of IoT systems with cloud-edge collaborative computing, named VSA-SD. This method solves the challenging problems of achieving highly efficient, robust, and trustworthy service discovery in the IoT environment by exploiting the advantages of VSA in task flow encoding and similarity measures. The main contributions of this paper are embodied in the following aspects:

1) A task flow encoding method based on VSA is designed and implemented, which can be used for general JSON workflow encoding.
2) A discovery mechanism through similarity measures and VSA-based task flow matching is designed and implemented, which is qualified for efficient, highly robust, and trustworthy distributed service discovery in IoT systems.
3) Through extensive experiments in real scenarios, it is verified that VSA-SD has higher performance in terms of accuracy, overhead, response delay, and other metrics than centralized, hybrid, and distributed service discovery schemes.

The rest of the paper is organized as follows. Section II gives a brief overview of the related work. Section III describes the basic principle of VSA. Section IV elaborates on the design and implementation of VSA-SD. Section V presents the experimental evaluation of VSA-SD. Section VI summarizes the whole paper and discuss the future work.

## II. Related Work

At present, in the commercial field, service discovery methods are mainly divided into two categories: one is the client-based service discovery model, and the other is a server-based service discovery model. The main difference is whether the client stores the service list information. In the client-based mode, if invoking a service, the service list must be first gotten from the service registry and then made a service call according to the caller's local load-balancing policy. In the server-based mode, the caller directly requests the service registry, which implements the service call using its load-balancing policy. In the latter mode, the caller does not need to maintain service discovery logic and service registration information on its nodes. Apache Zookeeper, HashiCorp Consul, and Netflix Eureka are popular service discovery frameworks today. Although these frameworks are very technically mature, it is still unsatisfactory to implement service discovery frameworks in the IoT domain to achieve efficient, highly robust, and trustworthy service discovery.

Additionally, researchers have done a lot of research work in this area. In general, centralized registries are used for service management and service discovery through query directories. For example, Albalas [9] et al. proposed a service discovery method using CoAP resource catalogs to maintain services in the network. The catalogs were updated to keep data freshness, and energy consumption was optimized by changing the update interval. Kim [10] et al. proposed a service discovery approach by registering DNS names and service lists of IoT devices with DNS servers, which allows users to automatically configure the devices and their services through a remote control system. Due to the uniqueness of the registry, these centralized service registration management approaches are short of scalability, and the entire IoT system will not function if the registry breaks down.

Thus, researchers have proposed hybrid multi-registry discovery approaches. For example, considering the large scale of physical services, Gomes [14] et al. proposed a semantic-based discovery service mechanism (QoDisco), which consists of a set of repositories that store resource descriptions according to an ontology-based information model and provides multi-attribute and range query capabilities. Cabrera [15] et al. proposed an adaptive service model supporting service discovery for smart

TABLE I
COMPARISON OF EXISTING SERVICE DISCOVERY MECHANISMS IN THE IOT ENVIRONMENT

| Service Discovery Mechanisms | | | Accuracy | Overhead | Response Delay | Robustness | Trustability | Interoperability | Mobility |
|---|---|---|---|---|---|---|---|---|---|
| Centralized | | [9] | ✓ | ✓✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| | | [10] | ✓ | ✓✓ | ✓ | ✓ | ✗ | ✗ | ✓ |
| Hybrid | | [14] | ✓✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓✓ |
| | | [15] | ✓✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Distributed | Unstructured | [16] | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| | | [17] | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ |
| | Structured | [18] | ✓✓ | ✓ | ✓✓ | ✓ | ✓ | ✓ | ✗ |
| | | [13] | ✓✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓✓ |
| | VSA-SD | | ✓✓✓ | ✓✓ | ✓✓ | ✓✓✓ | ✓✓ | ✓✓ | ✓✓✓ |

More ✓ means that the indicator performs better, ✗ means that it is not considered or not supported.

cities. The model organizes service information based on city events and introduces an adaptive architecture that tracks discovery metrics and moves information about the service between registries to maintain discovery efficiency. The hybrid multi-registry discovery model can effectively solve the problem of a centralized registry, the breakdown of which can prevent the whole system from working. However, keeping data in multiple registry centers leads to an enormous waste of resources. Furthermore, due to the dynamic availability and context awareness of entity services, this approach will bring huge update and maintenance overhead, so it will also impact the scalability and maintainability of the IoT system.

Hence, Xia [16] et al. proposed an efficient Social-Like Semantic-Aware (SLSA) service discovery mechanism, by mimicking human-like social behavior and exploring collective intelligence. The mechanism is capable of discovering desired services in a fast and scalable manner. Sharma [17] et al. considered distributed service discovery protocols for the connected vehicle environment, aiming to facilitate service discovery and service selection, and proposed a cluster-based service discovery approach to discover neighboring vehicles using neighbor awareness. Both of them are unstructured distributed service discovery. On the contrary, concerning structured distributed service discovery, Pereira [18] et al. proposed a distributed resource discovery architecture based on the MQTT protocol (MQTT-RD). The proposed architecture enables decentralized discovery and management of devices across multiple networks by introducing plug and play capabilities to devices. Santos [13] proposed a new P2P DHT-based resource discovery service in which fog nodes provide their computation resources as a service to mobile users. A DHT overlay is established between them to provide a decentralized and scalable discovery mechanism to the fog nodes. In this scheme, each mobile user selects the nearest fog node based on the relative delay and efficiently obtains the information to load and use the required service by querying the DHT coverage of the fog node.

Table I provides a comparative analysis of the existing research on service discovery in the IoT environment described above using the following metrics:

- *Accuracy:* indicates the difference between the discovered services and the expected discovered services.
- *Overhead:* indicates the maintenance and management overhead in service discovery.
- *Response Delay:* indicates the processing time to find the specified service.
- *Robustness:* indicates whether the correct service can be found with information errors occurring during service discovery.
- *Interoperability:* indicates whether an interoperable method exists to handle services that may differ in type, format, and technology.
- *Trustability:* indicates the trustworthiness of the results of service discovery.
- *Mobility:* indicates whether services can be continuously discovered when the corresponding devices dynamically move.

As can be seen from Table I, first, since the centralized scheme has only a single node registry to manage the service information, the overhead of maintenance management is low, but the service cannot be updated in time, so the accuracy of service discovery in a dynamic environment is low. In addition, the deployment of a single node leads to a strong dependence on the stability of that node, and if a node fails, the system cannot continue to operate. Finally, when performing service discovery, the overhead, and response delay increase significantly as the number of information increases. In the IoT domain with massive amounts of data, the centralized service discovery faces significant challenges, and other performance issues, such as robustness, are not adequately addressed. Although the hybrid multi-registry service discovery approach solves the single-node failure problem in centralized methods and achieves some robustness, it has similar response delay with the centralized approach. Both the structured and unstructured distributed service discovery methods still have some shortcomings, as the service discovery accuracy and response delay depends on the resource capacity of the working nodes. Therefore, they are not suitable for resource-constrained IoT nodes. In addition, the distributed service discovery methods based on DHT cannot meet the requirements of fuzzy search due to the principle of hash tables.

## III. PRINCIPLE OF VSA

VSA is a set of models created by psychologist Gayler [19] for representing and manipulating data in hyperdimensional spaces. The idea was initially proposed as an architecture by psychology and cognitive neuroscience as a connectionism for symbolic

model reasoning [20]. In VSA, information is represented by hyperdimensional vectors (HV) also called hypervector, and the encoded information is distributed over all components of the vector. VSA is a loss dimensionality reduction method that compresses a large amount of data into a fixed-size vector, thereby capturing associations between similarities and enabling classification to construct the data. For example, given a random hypervector $HV$ in a hyperdimensional space (typical dimensionality $N$=10,000), the vector provides many desirable properties, and the vast majority of vectors in the vector space are nearly orthogonal to the $HV$. In general, the vector space contains $2^N$ approximately orthogonal vectors, so each $HV$ can naturally represent semantically distinct objects, such as Arabic numerals [21]. Two hypervectors, $HV$ and $HV'$, with sufficiently high similarity can be deemed as related.

VSA is capable of supporting a variety of cognitive tasks, such as (a) semantic combination and matching; (b) representation of meaning and sequence; (c) analogical mapping; (d) logical reasoning. With VSA, they are highly adaptive to noise and can be used in distributed architectures [22]. As a result, it has been widely used in areas such as classification modeling, natural language processing (NLP), and cognitive modeling. Considering that computing hardware in the IoT system is resource constrained and prone to error, traditional distributed service discovery mechanisms mentioned above are hard to be implemented in practice. Due to the distributed service description and simple matching mechanism with hypervectors, VSA has low requirement on resource of individual node in system. Therefore, it is well-suited for designing mechanisms for IoT service discovery. Besides, using hyperdimensional computation, VSA achieves robustness and energy efficiency by allowing hardware-induced or transmission-induced errors with little or no performance loss [23].

Below is a list of concepts related to VSA to better understand the method presented in this paper.

*Hyperdimensional Space:* VSA is also called as hyperdimensional computing (HDC) [24] because it uses very high dimensions for representation and computation. Of course, the choice of dimensions is not deterministic and depends to some extent on the problem itself, but there are some simple empirical rules (e.g., $N$>5,000). In general, once the size is determined, it is fixed and cannot be changed. In addition, there are many types of hyperdimensional space, such as $\{0,1\}^N$, $\{-1,1\}^N$, and $\{-\pi,\pi\}^N$[25] etc. Thus the selection of an appropriate form of vector space and the determination of its dimensionality is of paramount importance when it comes to representing a specific data structure in a problem. These aspects are crucial in ensuring an accurate and efficient representation of the data.

*Blessing of Dimensionality:* Random vectors are generated by sampling each dimension independently and uniformly in hyperdimensional space. These randomly chosen vectors are nearly orthogonal to each other, which is a phenomenon known in mathematics as the concentration of measures. A characteristic of the phenomenon is that as the dimension of vectors increases this approximate or pseudo-orthogonal state converges to complete orthogonality. It is called the blessing of dimensionality in [26], as illustrated in Fig. 2.

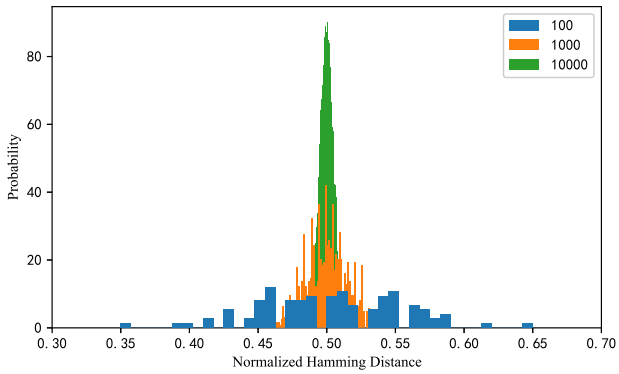


Fig. 2. Increasing probability of two random hypervectors orthogonal to each other (with normalized Hamming distance around 0.5), when the dimension of vectors $N$ increases from 100 to 10000.

*Similarity Measure:* The matching or inference in VSA is based on the similarity between the hypervectors. Many similarity measures have been proposed in VSA, such as dot product, cosine similarity, Hamming distance, and overlap [27]. Appropriate metrics should be chosen for different vector spaces. This paper uses Hamming distance, denoted $HammingDist(HV, HV')$, for measuring similarity.

*Atomic Vectors:* When designing a VSA model for problem-solving, a set of componential concepts and symbols are usually defined for depicting it. Generally, non-simplifiable and uniquely independent hypervectors, which are called seed vectors or atomic vectors [28], are designed as componential symbols. The other hypervectors in the VSA computation process can be unified as the combination of atomic vectors through mapping, encoding, projection, or embedding operations. Randomly generated vectors are generally used to assign atomic vectors.

*Vector Memory:* Atomic vectors are stored in content-addressable memory, either as matrices or associative memory [29], where a nearest-neighbor search can be taken to perform inference or matching [30]. This process is also known as cleanup memory.

VSA uses hypervectors for computation and has a set of vector operators, which are *Superposition*, *Binding*, and *Permutation*, which are described below in more detail.

*Superposition:* Superposition is the basic operation for combining multiple hyperdimensional vectors into a single related vector. In VSA, this operation is also known as bundling and addition. We use the term superposition in the paper. The simplest unconstrained superposition (denoted by $S$) is:

$$S = x \oplus y. \tag{1}$$

In some circumstances, the superposition operation may require custom settings. So it can be defined as

$$S = f\left(x \oplus y\right). \tag{2}$$

where $f(\cdot)$ is a function that restricts the value of $S$. For example, in the use of binary vector spaces, a suitable binary differential

threshold needs to be defined to ensure that the vectors obtained by superposition and combination are still in binary form. The operator can also be represented by other symbols, such as $+$, and multiple vectors are superposed together noted as $\sum$. An important feature of this superposition operation is that the combined hypervectors will be similar to each of the combined hypervectors, i.e., not orthogonal. So the superposition can be seen as an operation for unstructured, additional similarity preservation. As in the above example, $x$ and $y$ are unrelated atomic vectors, but the vector $S$ obtained by superposing will be similar to $x$ and $y$. However, the similarity will gradually decrease as more hypervectors are superposed together.

*Binding:* As mentioned before, the superposition operations alone can lead to information loss of the original objects, due to its associative nature, when the superposition operations are applied recursively. To represent the composite structure, a binding operation, also called multiplication, is used, which is an operation that maps two or more hypervectors to another hypervector. For example, the hypervector ($B$) representing the bound objects of $x$ and $y$ is:

$$B = x \otimes a \oplus y \otimes b. \tag{3}$$

The operator can also be represented by other symbols, such as $\odot$, and the binding of multiple vectors is denoted by $\prod$, which is reversible by the unbinding operation:

$$\begin{aligned} B \otimes a &= x \otimes a \otimes a \oplus y \otimes b \otimes a \\ &= x \oplus y \otimes b \otimes a \\ &= x \oplus noise \approx x. \end{aligned} \tag{4}$$

It should be noted that for the Exclusive OR (XOR) operation, its inverse operation is $a \otimes a = 0$, so $x \otimes a \otimes a = x$. The inverse operations of other operations will be different, but the main idea is the same. Moreover, when the unbinding operation is applied to the superposition of hypervectors, the result will be approximately equal to the initially binding hypervector, as shown in (4) because there will be a superposition of noise vectors $y \otimes b \otimes a$. An important feature of the binding operation is that the bound hypervectors will be dissimilar (pseudo-orthogonal) to each binding hypervector, since there are no similar vectors in the vector memory.

*Permutation:* It is an unitary operation, which is performed on one hypervector to obtain a dissimilar hypervector in regions of hypervector space that do not interfere with other representations. Similar to binding, it includes circular shifts, circular convolution, and Vector-derived Transformation Binding (VTB) [31], etc. However, unlike binding, the same permutation can be used recursively, with each round of recursion projecting into a previously unoccupied region. The vector $P$ obtained by permutation is:

$$P = \rho(x). \tag{5}$$

The permutation operation has the function of indicating the position of the object by the number of permutation operations. At the same time, the hypervector after permutation operations cannot be retrieval from the vector memory, thus ensuring its trustworthiness.

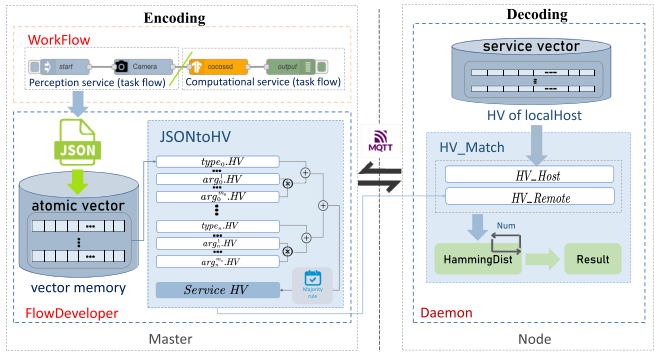


Fig. 3. Overview of VSA-SD.

## IV. VSA-SD Mechanism Design and Implementation

During the development of VSA, many different types of VSA implementations have been proposed [32]. The focus of this paper is not on optimizing existing VSA models but on designing and verifying the feasibility of VSAs for task flow discovery. Therefore, the traditional VSA model is used. In this section, we describe how to utilize VSA to implement task flow encoding as an efficient, highly robust, and trustworthy service discovery method.

An overview of the designed VSA-SD is illustrated in Fig. 3. It is an efficient service discovery solution using hyperdimensional computation. In the encoding phase, the perceptual modules in TaskFlow are transformed into JSON task flows in FlowDeveloper, and then the JSON task flows are parsed and encoded into service hyperdimensional vectors using specific operators (illustrated by the JSONtoHV module in Fig. 3, see Section IV-A for detail). In the decoding stage, i.e., service matching, the matching result is obtained by normalizing the Hamming distance calculation between the request hypervector ($HV\_Remote$) and the local hypervector ($HV\_Host$) (illustrated by the HV_Match module in Fig. 3, see Section IV-B for details). The system model and formulas are presented in this section.

Important symbols are listed in Table II.

### A. VSA-based JSON Task Flow Encoding (JSONtoHV)

In IoT systems, task requirements can be defined using a workflow approach as applications consisting of control and data dependencies and logical reasoning required for distributed execution. Workflows have been widely used to define and execute computational experiments, iterative observations, and simulated data [33]. Among them, workflows in JSON are more structured and can describe tasks clearly, which are easy to analyze. Therefore, this paper focuses on implementing VSA encoding for JSON task flows. Taking the perception service in NBUFlow [34] as an example, as shown in Fig. 4, it is described in JSON task flow.

First, the choice of hyperdimensional space and atomic vectors should be considered. Taking $N$=10,000 as the most commonly used space dimension in classical VSA architecture, $\{0,1\}^N$ is used as the vector space $\mathbb{V}$ due to the simplicity of

TABLE II
SYMBOLS USED IN THE PAPER

| Symbol | Definition |
|---|---|
| $N$ | dimensionality of hypervectors |
| $\mathbb{V}\epsilon\{0,1\}^N$ | 0 and 1 as elements in vector space in N-dimensional space |
| $HV$ | service hyperdimensional vectors |
| $Random()$ | random generation function of N-dimensional hypervectors in $\mathbb{V}$ |
| $type_i$ | type of the $i$th task module |
| $arg_i^j$ | the $j$th argument vector in the $i$th task module |
| $m_i$ | number of arguments in the $i$th task module |
| $\rho^i$ | $i$ permutation operations, which is inverse operation when $i$ is negative |
| $Num$ | number of superposition vectors required to compose the service vector |
| $C$ | vector memory |
| $HammingDist()$ | function of calculating normalized Hamming distance of two hypervectors |
| $ARG_i$ | vector after binding all arguments of the $i$th module |

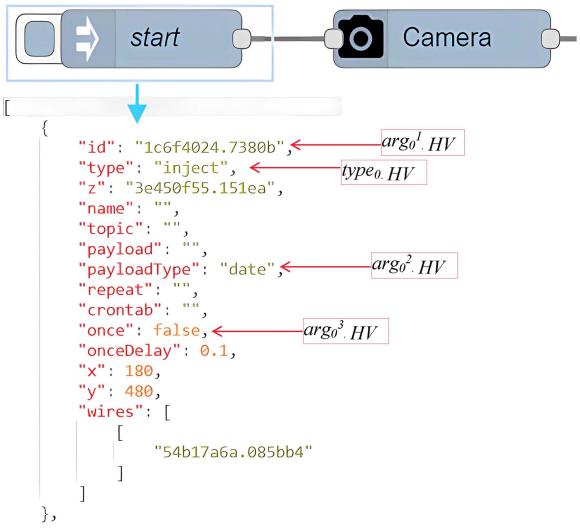


Fig. 4. JSON task flow description of a perception service in NBUFlow.

binary operations and storage, so that it can operate stably on devices with limited computing and storage resources. For the JSON task flow shown in Fig. 4, we decompose the keyword and value pairs into basic objects, such as 'type' and 'inject', etc. Note that the basic objects do not describe all elements, so they must be filtered by a filtering procedure to reduce the complexity of the task processing, such as the keywords $x$, $y$, $z$, etc. Next, random values are generated from the vector space of 10,000 dimensions with probability 1/2 for 0 and 1 to form the atomic hypervector. Once determined, the atomic vectors cannot change throughout the system's operational lifetime. The rest of service vectors are composed of atomic vectors. In an increasing number of tasks, we can continue to obtain vectors randomly without worrying about running out of independent vectors because time runs out before they run out [35].

Then, the element VSA operators should be determined. Depending on the application scenario, the operators may need to be changed.

For the *Superposition* operation, this paper uses a threshold function for binarization based on ordinary mathematical addition to ensure that the vector remains stable in binary form. In addition, we take the majority rule to set the threshold, i.e., half the number of superposition is used as the threshold. Of course, if an even number occurs, a random hypervector must be added at the end to get an odd number, thus breaking the tie and expressing it in the following form:

$$HV = \begin{cases} \sum_{i=1}^{n} HV_i, & n\%2 = 1 \\ \sum_{i=1}^{n} (HV_i) \oplus Random(), & n\%2 = 0 \end{cases}, \quad (6)$$

where $n$ denotes the number of superposition vectors, and $HV_i$ is the superposition service hypervector.

For the *Binding* operation, we use the XOR operation for binary operations, because the XOR operation can be built into memory, eliminating the need to move 10,000-dimensional vectors in and out of the Arithmetic Logic Unit (ALU). Due to the simplicity of binary operations, VSAs can be efficiently accelerated by hardware while reducing design complexity [36]. The specific form is:

$$B = \prod_{j=1}^{m_i} arg_i^j, \quad (7)$$

where $arg$ denotes the valid arguments in the task module, and its value corresponds to the value in the JSON task flow.

For the *Permutation* operation, a circular shift is used, thus reducing the consumption caused by using convolution or matrix operations on the device [30]. This operation is represented by a circular left shift and its inverse by a circular right shift. The specific expression is as follows:

$$P = \begin{cases} \rho^i(HV), & left\ shift \\ \rho^{-i}(HV), & right\ shift \end{cases}, \quad (8)$$

where the new vector $P$ after the operation is pseudo-orthogonal to the original vector $HV$ to avoid being matched.

For workflows constructed with JSON, the hypervectors are generated by the operators in the VSA described above:

$$J = \sum_{i=0}^{n} \rho^{2i} \left( type_i \oplus \rho \prod_{j=1}^{m_i} arg_i^j \right). \quad (9)$$

The task flow represented by $J$ is a 10,000-dimensional combined hypervector. It facilitates fuzzy matching of services by separating the keyword $type$ from other arguments $arg$ (as

shown in Fig. 4), since in some IoT systems service discovery does not necessarily require exact matching. For example, to discover the service module for Capturing Image, the image size may be taken as a parameter, which cannot be exactly appointed. Therefore, fuzzy matching is achieved by separating the task module $type$ from the $arg$ in the JSON task flows. If an $arg$ is inconsistent or changes, the task flow cannot be successfully matched accurately. But if the $type$ matching is successful, the fuzzy matching is completed. The specific expansion of (9) is as follows:

$$J = type_0 \oplus \rho\left(arg_0^1 \otimes arg_0^2 \cdots\right) \oplus \rho^2\left(type_1\right) \oplus \rho^3\left(arg_1^1 \otimes arg_1^2 \cdots\right) \oplus \cdots . \tag{10}$$

Here, the vector $J$ does not have a corresponding atomic vector in the vector memory ($C$), but the first element, $type_0$, is exposed and unaffected by the binding and permutation operations, while the other vectors are hidden and protected by the permutation operations. The portion of the vector that is affected by the binding and permutation operations is considered as noise.

Although the vector $type_0$ has undergone a partial superposition operation, which causes it to lose some of its properties and to be partially different from the atomic vector $type_0$ in vector memory, it is still closer to its original vector than any other vectors in space. Moreover, the vector protected by the permutation operation is exposed using its inverse operation, as follows:

$$\rho^{-1}(J) = \rho^{-1}\left(\sum_{i=0}^{n} \rho^{2i}\left(type_i \oplus \rho \prod_{j=1}^{m_i} arg_i^j\right)\right). \tag{11}$$

It can be expanded as follows:

$$J^{-1} = \rho^{-1}\left(type_0\right) \oplus \left(arg_0^1 \otimes arg_0^2 \cdots\right) \oplus \rho\left(type_1\right) \oplus \rho^2\left(arg_1^1 \otimes arg_1^2 \cdots\right) \oplus \cdots . \tag{12}$$

At this point, $type_0$ is protected as noise by the inverse permutation operation $\rho^{-1}$. $arg_0^1 \otimes arg_0^2 \cdots$ is exposed by the inverse permutation operation, but because the exposed argument vector is not a single atomic vector, it is a composite vector formed by binding. Due to the nature of the binding, the generated vectors are not similar to the original vectors and thus cannot be matched for detection in vector memory. The traditional VSA mechanism is to unbind objects by discovering their internal information, while this study applies it to task discovery, which only requires similarity matching. The following section will focus on explaining how it is used differently from the traditional VSA mechanism.

As shown in Algorithm 1, the input of the program is a preprocessed JSON workflow. First, the keywords (i.e.,the pair of keyword and value in the JSON task flow) are mapped to an atomic vector by generating a random $N$-dimensional vector of integers in space $\mathbb{V}\epsilon\{0,1\}^N$ as an atomic vector. After that, it does not and cannot change, and it must be consistent throughout the system lifecycle and stored in vector memory $C$ (Line 2-6 in Algorithm 1).

Then, the atomic vectors are combined using the VSA operator to form service vectors. First, the task flow are divided into several modules according to the number of $type$ in JSON. Each of these modules contains two parts, i.e., the value corresponding to the $type$ keyword and the value corresponding to other valid keywords, which are called as the argument $arg$. Next, the arguments in each module other than $type$ are bound, i.e., using the XOR operation (Line 8 in Algorithm 1). Second, considering the process of forming service vectors, if all the service vectors are superposed directly into the service hyperdimensional vector $HV$, it will cause the problem of multiple vectors being exposed at the same time. It will be impossible to find the corresponding atomic vector in the vector memory, which will lead to the discovery of untrustworthy services. Therefore, multiple permutation operations (Line 9-11 in Algorithm 1) are required to achieve the effect of hiding vectors. Subsequently, based

**Algorithm 1:** JSONtoHV.

**Input:** A perceptual service in the form of JSON task flow.
**Output:** A service hyperdimensional vector $HV$ and the number of superposition vectors $Num$ required to compose $HV$.

1: **Initialize:** $N = 10{,}000$;
$\mathbb{V}\epsilon\{0,1\}^N$;
$C =$ Null;
$HV =$ Null;
$rand = Random()$;
$Num = 0$.
// **Step 1.** Generate atomic vectors for keywords in the JSON task flow
2: **for** $i = 0$ to JSON.keyword.number **do**
3: **if** JSON.keyword $\notin C$ **then**
4: $C_i \leftarrow Random()$
5: **end if**
6: **end for**
// **Step 2.** Combine atomic vectors into service vectors
7: **for** $i = 0$ to JSON.type.number **do**
// Complete the XOR binding operation for valid arguments in each module
8: $ARG_i \leftarrow C.arg_i^1\ XOR\ C.arg_i^2 XOR \cdots C.arg_i^{type.arg.number}$
// Circular left shift by one position
9: $PARG_i \leftarrow (ARG_i \gg (N-1))|(ARG_i \ll 1)$
10: $HV_i \leftarrow((C.type_i \oplus PARG_i) \gg (N-2i))| ((C.type_i \oplus PARG_i) \ll 2i)$
11: $HV \leftarrow HV \oplus HV_i$
12: $Num \leftarrow Num + 2$
13: **end for**
// **Step 3.** Majority addition voting for binarization
14: $HV \leftarrow HV \oplus rand$
15: **for** $j = 0$ to $N$-1 **do**
16: **if** $HV[j] <$ ($Num$+1)/2 **then**
17: $HV[j] \leftarrow 0$
18: **else**
19: $HV[j] \leftarrow 1$
20: **end if**
21: **end for**

on the consideration of the inverse permutation operation, the number of permutation shifts is set to $2i$ so that after the inverse operation, it is still guaranteed that only the unique features of the vector are exposed (as shown in (12)). Finally, the service vector $HV$ is binarized using majority voting. Since each module is divided into $type$ and $arg$ when coding and designing the modules, the superposition number $Num$ is always even, so a random vector must be added to make $HV$ in odd state (Line 14 in Algorithm 1). The distribution of each feature element in the task is represented by the JSONtoHV algorithm, and the final result is a 10,000-dimensional vector $HV$ and the number of superposition vectors $Num$.

### B. VSA-Based Task Flow Matching (HV_Match)

As described above, through Algorithm 1, the workflow is transformed into a hyperdimensional vector by VSA encoding, and stored as service vector in local devices, as shown in Fig. 4. As a way of representing service description, it allows semantic comparisons to be made in resource constraint devices in a given environment. Efficient service discovery in a complex space is performed by sending semantically rich queries in the form of vectors to the network. In IoT systems, it is required that each working node or gateway contains a component or program capable of processing vectors, i.e., capable of performing vector reception and matching. The traditional VSA approach is to parse and expose vectors and then discover the elements by querying the vector memory [37]. Differing from the traditional approach, VSA-SD does not discover vectors by querying them, instead, it uses matching based on the properties of hyperdimensional vectors. When two vectors have the same properties, they are seen as similar. Therefore, we use VSA to represent vectors in a distributed way and perform the exposure matching one by one. Although there will be much noise, two similar vectors will still be closer to each other than any other vectors in space.

In order to achieve the match, the requested vector (denoted as $HV\_Remote$ in Fig. 4) is constructed in a slightly different way than the vector encoded in the previous section, as follows:

$$J = \sum_{i=0}^{n} \rho^{4i} \left( type_i \oplus \rho^2 \prod_{j=1}^{m_i} arg_i^j \right). \tag{13}$$

The purpose of the different number of times used on the vector permutation operation, in this case, is to prevent the permutation operation of the same structure when exposing the features, resulting in all the permuted hidden features being matched, as follows:

$$J = a \oplus \rho b \oplus \rho^2 c \oplus \rho^3 d;$$
$$J' = A \oplus \rho b \oplus \rho^2 c \oplus \rho^3 d. \tag{14}$$

Although the features of $a$ and $A$ are exposed, and the permutation operation protects the superposition, a large number of protected vectors, i.e., $\rho b \oplus \rho^2 c \oplus \rho^3 d$, are matched under the same rule, resulting in the exposed $a$ being incorrectly judged to be matched with $A$. Then the inverse permutation operation is performed to remove the subsequent protection, and finally matched task flow module is obtained. This is not what we expect. Therefore, we use different rules for the initial service vector, and the following results are obtained using (13):

$$J' = A \oplus \rho^2 b \oplus \rho^4 c \oplus \rho^6 d. \tag{15}$$

At this point, the protection rule is modified so that $\rho^2 b \oplus \rho^4 c \oplus \rho^6 d$ will not be similar to $\rho b \oplus \rho^2 c \oplus \rho^3 d$, and the above error will not occur. Even though there may be an identical situation at one bit, it will not result in matching of all features. So the final matching result is not affected. The specific implementation of VSA-based task flow matching mechanism ($HV\_Match$) is as follows:

Before the formal deployment, working nodes receive request messages sent remotely by a custom functional component. The system must deploy and install this component or program at the

**Algorithm 2:** HV_Match.

**Input:** Task vector deployed in local node $HV\_Host$;
Request vector sent remotely $HV\_Remote$;
Number of superposition operations for service vectors deployed in local nodes $HV\_Host.Num$;
Number of superposition operations for service vectors sent from remote nodes $HV\_Remote.Num$.
**Output:** Matching result $id$ // 0 means failure, 1 means exact match, 2 means fuzzy match.

1: **Initialize:** $id, j, k = 0$;
//Set matching threshold
$Threshold = 0.48$.
// **Step 1.** Quickly determine matching
2: **if** $HV\_Host.Num$ != $HV\_Remote.Num$ **then**
3: **return** $id$
4: **else**
// **Step 2.** Decode the matches one by one
5: $Num = HV\_Remote.Num$
6: **for** $i$=1 to $Num$ **do**
7: **if** $(HammingDist(HV\_Host, HV\_Remote) < N * Threshold)$ **then**
8: Circular right shift $HV\_Host$ by one position
9: Circular right shift $HV\_Remote$ by two positions
10: **if** $i$ is odd **then**
//Record odd-bit success
11: $j++$
12: **else**
13: $k++$
//Record even-bit success
14: **end if**
15: **end if**
16: **end for**
// **Step 3.** Judgment of matching results
17: **if** $j + k == Num$ **then**
18: $id = 1$
19: **else if** $j == Num/2$ **then**
20: $id = 2$
21: **end if**
22: **end if**
23: **return** $id$

time each node joins the system to ensure the stable operation of the discovery mechanism.

First, the HV_Match component holds waiting for service requests via a specific protocol. It receives a request message from the remote Master node containing a hyperdimensional vector ($HV\_Remote$) and the number of superpositions ($HV\_Remote.Num$). At the same time, the hyperdimensional vector of the task it is running ($HV\_Host$) and its superposition number ($HV\_Host.Num$) is read from the current node. The matching threshold is set to 0.48, reason of which is explained in Section V-C1. After initialization, $HV\_Match$ quickly determines whether the number of service modules superposed when constructing $HV\_Host$ and $HV\_Remote$, are the same (Line 2 in Algorithm 2). Here we set that hyperdimensional vectors superposed by different number of modules cannot be matched because the encoded task flows will deviate from the user's expectation to some extent by adding or reducing modules. Of course, it can also be modified according to the specific scenarios under different task descriptions.

Then, if $Num$ is consistent to $HV\_Host.Num$, the matching process starts. The calculation is performed using the normalized Hamming distance (Line 6 in Algorithm 2), where the threshold is set to the dimensionality of $HV$s multiplied by 0.48 when the discovery error probability is one in a billion [35]. The specific threshold can be adjusted according to different scenarios. The computation (i.e., the inverse permutation operation) is taken sequentially according to the number of superpositions. The odd and even cases are separated to refine the matching results. If all are successful, the exact match is complete. If only the odd cases are matched successfully, and the even cases are not matched exactly, it means that only the module types are matched successfully, but the internal arguments are inconsistent. In this case, fuzzy matching is successful. Otherwise, it means that the match fails, i.e., the $id$ is 0 (Line 17-23 in Algorithm 2). At this point, no information is returned to the requesting party to reduce communication overhead.

## V. Experimental Evaluation

### A. NBUFlow-Based Experimental Validation

In this section, the above scheme is implemented in our established IoT task construction and offloading test platform NBUFlow [34], which is a cloud-native technology-based system validation platform using K8s/Docker/Node-RED, and the system architecture is shown in Fig. 5. In this platform, we modify the FlowDeveloper module running on the Master node and the Daemon module running on the work node to embed the JSONtoHV algorithm and the HV_Match algorithm, respectively. We verify the performance of the method proposed in this paper in terms of accuracy, overhead, response delay, and other metrics.

*Embedding the JSONtoHV Algorithm Into the FlowDeveloper Module:* The FlowDeveloper module is based on the Node-RED task flow method to provide users with a visual interface to define tasks and achieve the task partitioning function. As mentioned above, the functional modules of an IoT task can be divided into perception service and computation service. For example, in the

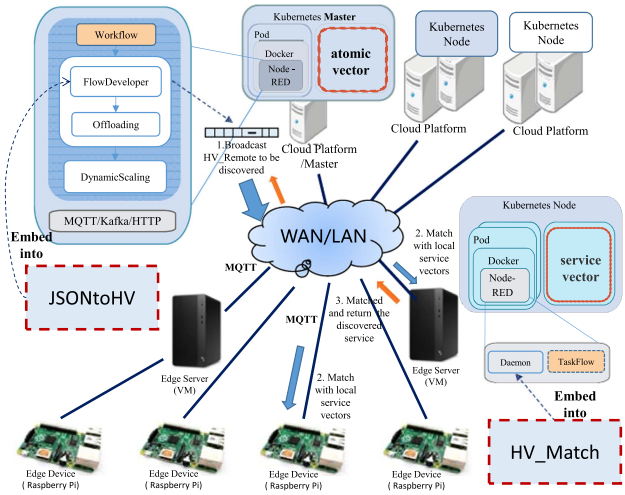


Fig. 5. Experimental system architecture.

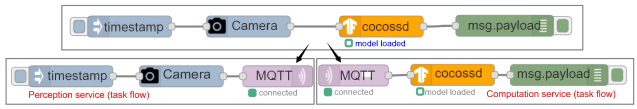


Fig. 6. Example of task division.

video surveillance task shown in Fig. 6, the $Camera$ module represents the perception service, and the $cocossd$ module represents the computation service in the task flow. Communication between the two subtasks after partitioning is done by adding $MQTT$ modules. The JSONtoHV algorithm is embedded in the FlowDeveloper module to implement the encoding function for the perception service, and the computation service is offloaded to a suitable cloud or edge server by the Offloading module (not the work of this paper).

*Embedding the HV_Match Algorithm Into the Daemon Module:* Daemon module can automatically receive task commands from Master nodes to deploy tasks automatically. After adding the HV_Match algorithm, the Daemon module can match task flows while keeping the automatic service deployment function. It is worth noting that during the initial deployment, both the cloud platform and the edge nodes in the system are idle and do not perform any perception or computation tasks, so the discovery mechanism does not work. When receiving an offloaded module, the node will distinguish the forms of the received data. For the JSON task flow, it will automatically parse and deploy it, while for the $HV$, it will match it against the local vector and return the result to the Master.

### B. Experimental Environment and Evaluation Metrics

To evaluate the performance of VSA-SD, two virtual hosts were first created using a Huawei TaiShan 200 server as the master node of the platform (CPU: 128x Kunpeng 920 @2.6 GHz, memory: 188 G, disk storage: 16 T) to achieve dual master backup. Then 2 Altas 200 DK are used as edge servers (CPU: 8x Arm Cortex-A55 @1.6 GHz, NPU: Ascend310, memory:

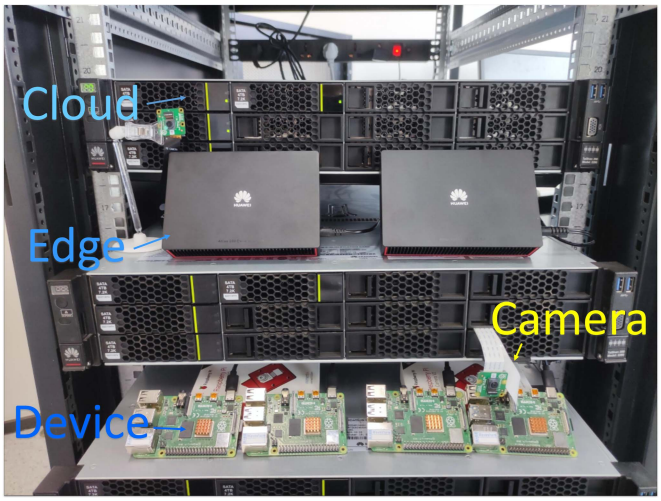


Fig. 7. Test platform built based on Huawei TaiShan server (Cloud), Atlas 200 (Edge), and Raspberry Pi4B (Device).

8 G, disk storage: 64 G). Finally, 4 Raspberry Pi 4B are used as device layer nodes (Pi) (CPU: 4x Arm Cortex-A72 @1.5 GHz, memory: 4 G, disk storage: 16 G), using a WiFi network to interconnect these platforms.

In addition, the experiment initializes each Raspberry Pi node and edge server node with a camera device and deploy a perception service to capture images, as shown in Fig. 7. One of the Raspberry Pi nodes and the edge servers are configured with the same parameters for the image capture task.

In this section, we use the metrics mentioned in the related work section, i.e., accuracy, overhead, and response delay, to evaluate the efficiency of the discovery mechanism. For the robustness of the discovery mechanism, it is judged in terms of whether the service can be accurately located in case of errors in service description. We also qualitatively analyze the other three metrics of the proposed mechanism, which are interoperability, mobility, and trustworthiness.

### C. Experimental Results

*1) Accuracy:* The accuracy of VSA-SD is verified by discovering the perception service module in video surveillance task shown in Fig. 6 as an example. The same perception service module is deployed in all edge servers and Pi nodes. A time of experiment starts by sending a discovery request from the master node. The first responder is selected as the service node when matching the required service. In the experiments, if two results are obtained when matching the same service from a Pi node and an edge server respectively, because the response from the edge server is faster than that from the Pi, the matching result will be obtained from the edge server. We repeat the experiments for 1,000 times with task flows of different complexity to obtain their corresponding normalized Hamming distances. Specifically, the number of Node-RED service modules combined together as a task flow is varied for different task complexity, which is denoted by $Num$ in experiments. The results of normalized Hamming distance for matching and mismatching with different task complexity are shown in Fig. 8. A successful match is indicated by the normalized distance between two vectors below a certain threshold. It is easy to see from the experiments that when setting the threshold to 0.48 it has a better matching effect. Even when the task complexity is 5, an almost 100% matching success rate can still be achieved. Moreover, it can be seen from Figs. 9 and 10 that even when the task complexity changes, VSA-SD achieves excellent stability with basically no large fluctuations and an average standard deviation of 0.024.

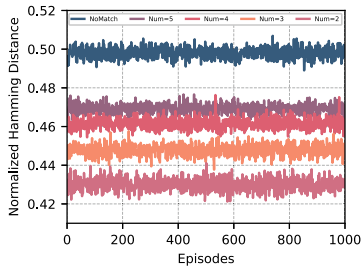


Fig. 8. Normalized Hamming distance for matching and mismatching with different task complexity.

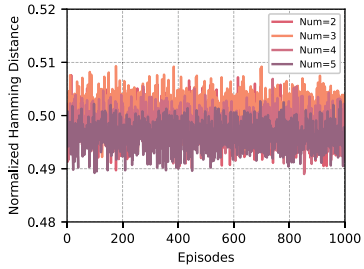


Fig. 9. Normalized Hamming distance for different vectors with different task complexity.

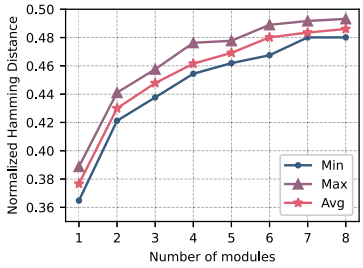


Fig. 10. Maximum, minimum, and average values of normalized Hamming distance for different number of modules.

*2) Overhead:* The service maintenance and management overhead of VSA-SD can be induced from the involving work in coding and storing atomic hypervectors in the master nodes, and maintaining the current service hypervector in the working nodes. Since the hyperdimensional vectors are stored in binary form, the maintenance and management overhead is greatly reduced. In addition, the storage overhead is quantified as a comparison metric in the experiments, for comparing the four approaches, which are centralized, hybrid, distributed, and VSA-SD. The experiments use the JSON task flow in NBUFlow as

TABLE III
COMPARISON OF MAINTENANCE AND MANAGEMENT OVERHEAD FOR DIFFERENT DISCOVERY APPROACHES

| | Native Services | Centralized | Hybrid | Distributed | VSA-SD |
|---|---|---|---|---|---|
| Initialization | $ini$ | $ini$ | $ini * N^+$ | $ini * \alpha * N^+$ | $ini * \beta$ |
| Add $M$ services | $ini + M$ | $ini + M$ | $(ini + M) * N^+$ | $(ini * \alpha + \varepsilon) * N^+$ | $ini * \beta + \phi$ |

TABLE IV
INITIALIZED VALUES OF THE EXPERIMENTAL PARAMETERS

| Parameter | Meaning | Value | Unit |
|---|---|---|---|
| $ini$ | initial storage cost of a service (i.e., native JSON task flow) | 4.22 | KB |
| $N^+$ | number of working nodes | [5,25] | / |
| $\alpha$ | the amount of space taken up by JSON task flow after converting it to a hashed table | 0.5 | KB |
| $\beta$ | the ratio of storage cost of the atomic vectors making up the service vector to $ini$ | 3.47 | / |
| $M$ | number of services added | 2 | / |
| $\varepsilon$ | increment of space occupied by the hash table with the addition of $M$ services | 0.1 | KB |
| $\phi$ | space occupied by the newly added atomic vectors of the VSA-SD system with the addition of $M$ services | [2.44,9.76] | KB |

the native description of the task, whose visualized description is shown in Fig. 6, and its storage overhead is expressed as $ini$. As shown in Table III, during the initialization stage of task deployment, the centralized approach has nearly the same storage overhead as $ini$ (ignoring other factors such as storing the $IP$ addresses of additional nodes). With rapid increase of nodes in the IoT systems, for the hybrid and the distributed approaches, their storage overhead will increase with the number of working nodes (denoted as $N^+$). For VSA-SD, the overhead of generating all atomic vectors composing the service hypervectors will be greater than that for the centralized approach. But as the system runs, the maintenance and management overhead of the centralized approach increases by almost as much as the number of services added (denoted as $M$). Because service hypervectors are composed of atomic vectors in VSA-SD, the new services only need to find the existing atomic vectors from the vector memory without adding too much additional overhead, so the storage overhead tends to stabilize after the initialization stage.

For convenience of comparison, the experimental parameters are idealized, and their values are initialized as shown in Table IV, where the initial service, i.e., the original JSON workflow, occupies 4.22 KB of space and consists of 12 atomic vectors, each of which is about 1.22 KB, so $\beta$ is 1.22*12/4.22. Next, the value of $N^+$ is initially 5, i.e., 5 working nodes, and each time 5 nodes are added. Correspondingly, 2 new services are added, and the value of $\phi$ is decremented from 9.76 to 2.44, each time by 2.44. Correspondingly, the number of additional atomic vectors decreases from 4 to 2 after each service is added. It should be noted that the additional services occupy the same storage space, but differ in the types of atomic vectors comprising the original services. $\alpha$ and $\varepsilon$ are set to fixed values in experiments. It is worth noting that setting them to the other values will yield different results, but the general trend of overhead changing is the same.

Fig. 11 shows the storage overheads of the four methods. We can see that the centralized, hybrid, and distributed service discovery methods show a significant increase in overhead as the number of services increases. While VSA-SD has a higher overhead when the number of services is small, due to the atomic vector combination mechanism, it consumes much less storage space with increasing number of working nodes and newly added services. However, the storage cost for VSA-SD gradually converges to a stable value.

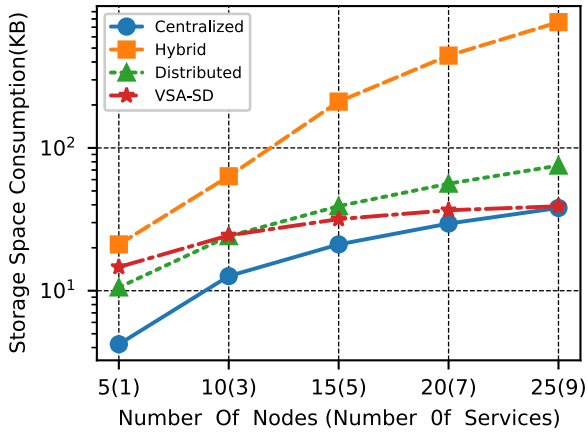


Fig. 11. Comparison of storage overhead of different methods.

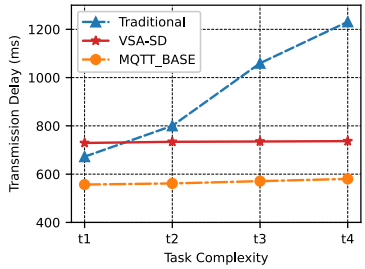


Fig. 12. Comparison of transmission delay between the traditional service discovery mechanism and VSA-SD with MQTT as a baseline for different task complexity.

*3) Response Delay:* Regarding the response delay, it is mainly divided into transmission delay, computation delay, and delay caused by other factors. Comparison of transmission delay for different service discovery mechanisms is shown in Fig. 12. Although transmitting discovery requests leads to an increase in transmission delay due to excessive dimensionality

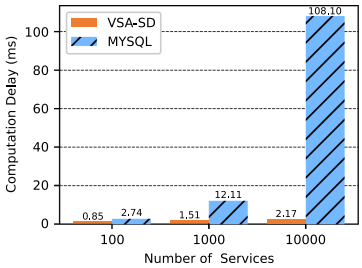


Fig. 13. Comparison of computational delay for discovering services with increasing number of services,in platform with 5 working nodes and 1 master node.

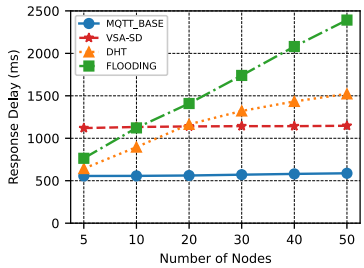


Fig. 14. Comparison of response delay between VSA-SD and other service discovery methods with different number of nodes, each of which provides a service.

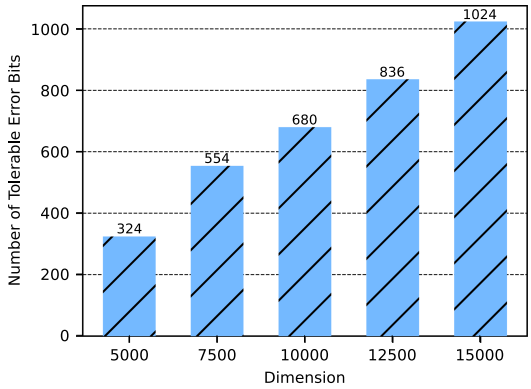


Fig. 15. Number of error bits tolerated by VSA-SD in different dimensions.

of hypervectors, VSA-SD can always maintain a stable transmission delay when the number of requests gradually increases, because it will combine them together and transmit them as a hypervector of fixed dimensionality. In contrast, the traditional accurate discovery approach needs to transmit native task flows or their description information, which causes the transmission time to increase as the task complexity (i.e., the number of task flows) increases. The transmission delay for MQTT_BASE is the one-round-trip time of sending requests from the source node to the registry node and sending discovery results back, which is always lower than VSA-SD.

Then, using the classic MySQL database for storing service data based on the registry as a comparison object, we conducted experiments with 100, 1,000, and 10,000 service data randomly generated, queried, and tested in turn. The results are shown in Fig. 13, where the computation delay increases rapidly as the number of services grows, in the case of database-based query. In the IoT environment, due to a large number of services and their dynamics, in the traditional registry-based approach, whenever any service data change, the database needs to update. Therefore, the frequent operations on the database lead to increased computation overhead and are challenging to manage and maintain it at a later stage. Other more powerful information management methods currently used by popular service discovery frameworks, such as Zookeeper, Netflix Eureka etc., also have the similar issues. When the number of services increases by the same amount, the computation delay of VSA-SD is much less than that of querying the database, because VSA-SD achieves service discovery using vector matching rather than querying the database. The larger the number of services, the more pronounced the effect.

Then, we conducted experiments to compare response delay of different service discovery schemes with the MQTT protocol taken as the baseline. Flooding and DHT were taken as representative schemes of unstructured distributed and structured distributed service discovery respectively, and compared them with VSA-SD in the same environment. Specifically, for the Flooding service discovery, instead of making nodes broadcast the registration information of provided services, it needs consuming nodes to send a request for the desired service, match the requested service through flooding routing, and return a response. The number of path hops that each request traverses in this experiment is related with the number of nodes whose locations were randomly generated, and the results obtained are averaged after removing outliers, as shown in Fig. 14. Since all the service description must be transmitted in each hop, the response delay is initially high and increases rapidly as the number of nodes increases. Next, for the structured distributed service discovery based on DHT, the service information is managed using the classical Chord algorithm and a structured distributed consistent hash lookup service is implemented. Its response delay is also very close to the theoretical average number of routing hops $(\log_2 N)/2$ [38]. The consistent hash operation eliminates the transmission cost, but it needs to query the hash table stored in multiple nodes, which induces route forwarding and causes some delay. So the approach based on DHT can maintain better performance with less number of nodes. Finally, for VSA-SD, because the query information transmitted from the master node is a binary vector of fixed dimension, directly to the working nodes in the system, so the response delay keeps stable with increasing number of nodes.

*4) Robustness:* Next, we evaluated the robustness of the VSA-SD by using vectors of different dimensions to represent the task flows. The task complexity in the experiments is set to 2, which means two task flows are combined together to be encoded as a hybervector. The dimension $N$ is set from 5,000 to 15,000, with the step of 2,500. In the experiment, 48% of the dimension was used as the threshold benchmark, and the difference between the Hamming distance and the corresponding threshold was evaluated for each dimension. The result is averaged over 1,000 times of experiments for each dimension. The results are shown in Fig. 15, which shows that even if some bits in the requested

service hypervector occur errors during transmission, a correct match can still be obtained in the end. Particularly, even with hundreds of bits of errors, VSA-SD can still correctly discover the service as long as the error rate does not exceed the specified threshold (i.e., 48% of the dimension in the experiments).

*5) Trustability, Interoperability, and Mobility:*

- *Trustability:* The service (task flow) is described as hyperdimensional vector in the VSA mechanism, where atomic vectors, encoding rules, and random vectors are hidden. So even if the encoded task flow is intercepted, the interceptor can only get a string of binary data and cannot obtain information about the task flow itself. Therefore, the VSA-SD service discovery mechanism naturally is secure, thus ensuring that the discovered service is trustworthy.
- *Interoperability:* In the experiments, Huawei TaiShan Server, Atlas 200, and Raspberry Pi 4B are used as experimental nodes. Although these platforms have different architectures, the atomic vectors and service vectors of VSA-SD can be stored in these nodes. VSA-SD runs in the binary system and is not constrained by the differences in platform architecture and running environment, and can run stably in platforms consisting of heterogeneous devices.
- *Mobility:* In the experiments, we made location (i.e., moving from one device to another) and status (i.e., switching between online and offline) of some deployed perception services change, while keeping the properties of these services unchanged. For the traditional registry-based approach, it is necessary to update the corresponding entry in the database. However, in the distributed approach, such as DHT based approach, it is necessary to update the hash table, which will incur much overhead in the highly dynamic scenarios of IoT. As for VSA-SD, because information about services in the system is stored in binary form in each working node, it will not change when the service location or status changes. Hence, the movement and status change of service in VSA-SD is transparent to the requester (master node), because the node who achieves successful service matching will return the discovery result.

## VI. Conclusion

It is significantly important to discover accessible perception services deployed by dedicated IT infrastructure service providers to reduce additional overhead caused by redundant deployment of perception services, so as to construct customized IoT system with low cost. Due to the shortage of existing service discovery solutions in meeting the requirements of efficiency, robustness, and trustworthiness in IoT scenarios with cloud, edge and end devices, this paper proposes a distributed service discovery method based on Vector Symbolic Architecture (VSA-SD). We first design and implement VSA encoding for perception services, which are JSON task flows in this paper. Then, we store the encoded services in a distributed manner using hyperdimensional vectors and measure the degree of matching between the required services and the stored services by calculating the Hamming distance to achieve service discovery. Finally, real-platform experiments demonstrate that VSA-SD outperforms centralized, hybrid, and distributed service discovery in the scenario with different task complexity, different number of services, and different number of nodes in terms of accuracy, overhead, and delay. The effects of error bits on the performance of VSA-SD are evaluated, and its performance in trustability, interoperability, and mobility is analyzed.

It should be noted that VSA-SD was presented in this paper as an approach of discovering perception services described with JSON task flows. It can be easily extended to be used as discovery mechanisms of computational services and other types of services in IoT systems. In future work, we plan to further optimize the discovery efficiency of VSA-SD, for example, by investigating intelligent algorithms of choosing dimensionality and threshold for trading off between overhead and robustness. We also plan to extend the scheme to support description and discovery of other task flows, and combine it with stream processing engine [39] to achieve data service of high quality from the IoT systems. Finally, we will open-source the optimized scheme and hope that more researchers will participate in this project to jointly promote the development of IoT service discovery and the research in the VSA domain.

## References

[1] P. Bellini, P. Nesi, and G. Pantaleo, "IoT-enabled smart cities: A review of concepts, frameworks and key technologies," *Appl. Sci.*, vol. 12, no. 3, 2022, Art. no. 1607.

[2] "The mobile economy," 2022. [Online]. Available: https://data.gsmaintelligence.com/api-web/v2/research-file-download?id=74384059&file=270223-The-Mobile-Economy-2023.pdf

[3] G. Tricomi, Z. Benomar, F. Aragona, G. Merlino, F. Longo, and A. Puliafito, "A nodered-based dashboard to deploy pipelines on top of IoT infrastructure," in *Proc. IEEE Int. Conf. Smart Comput.*, 2020, pp. 122–129.

[4] Y. Tu, H. Chen, L. Yan, and X. Zhou, "Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT," *Future Internet*, vol. 14, no. 2, 2022, Art. no. 30.

[5] M. Achir, A. Abdelli, L. Mokdad, and J. Benothman, "Service discovery and selection in IoT: A survey and a taxonomy," *J. Netw. Comput. Appl.*, vol. 200, 2022, Art. no. 103331.

[6] H. Chen, W. Qin, and L. Wang, "Task partitioning and offloading in IoT cloud-edge collaborative computing frame: A survey," *J. Cloud Comput.: Adv. Syst. Appl.*, vol. 11, 2022, Art. no. 86.

[7] M. Antunes, J. Quevedo, D. Gomes, and R. L. Aguiar, "Improve contextual IoT service discovery with semantic models," in *Proc. 9th Int. Conf. Future Internet Things Cloud*, 2022, pp. 227–233.

[8] B. Pourghebleh, V. Hayyolalam, and A. A. Anvigh, "Service discovery in the Internet of Things: Review of current trends and research challenges," *Wireless Netw.*, vol. 26, no. 7, pp. 5371–5391, 2020.

[9] F. Albalas, W. Mardini, and M. Al-Soud, "AFT: Adaptive fibonacci-based tuning protocol for service and resource discovery in the Internet of Things," in *Proc. IEEE 2nd Int. Conf. Fog Mobile Edge Comput.*, 2017, pp. 177–182.

[10] S. Kim, K. Lee, and J. P. Jeong, "DNS naming services for service discovery and remote control for internet-of-things devices," in *Proc. Int. Conf. Inf. Commun. Technol. Convergence*, 2017, pp. 1156–1161.

[11] S. Mastorakis and A. Mtibaa, "Towards service discovery and invocation in data-centric edge networks," in *Proc. 27th IEEE Int. Conf. Netw. Protoc.*, 2019, pp. 1–6.

[12] G. Tanganelli, C. Vallati, and E. Mingozzi, "Edge-centric distributed discovery and access in the Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 425–438, Feb. 2018.

[13] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards dynamic fog resource provisioning for smart city applications," in *Proc. 14th Int. Conf. Netw. Service Manage.*, 2018, pp. 290–294.

[14] P. Gomes et al., "A semantic-based discovery service for the Internet of Things," *J. Internet Serv. Appl.*, vol. 10, no. 1, pp. 1–14, 2019.
[15] C. Cabrera and S. Clarke, "A self-adaptive service discovery model for smart cities," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 386–399, Jan./Feb. 2022.
[16] H. Xia, C.-Q. Hu, F. Xiao, X.-G. Cheng, and Z.-K. Pan, "An efficient social-like semantic-aware service discovery mechanism for large-scale Internet of Things," *Comput. Netw.*, vol. 152, pp. 210–220, 2019.
[17] N. Sharma, N. Chauhan, and N. Chand, "Cluster based distributed service discovery in internet of vehicle," *J. Commun. Softw. Syst.*, vol. 17, no. 3, pp. 281–288, 2021.
[18] E. M. Pereira, R. Pinto, J. P. C. dos Reis, and G. Gonçalves, "MQTT-RD: A MQTT based resource discovery for machine to machine communication," in *Proc. IoTBDS*, 2019, pp. 115–124.
[19] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," 2004, *arXiv:cs/0412059*.
[20] T. Plate, "Distributed representations and nested compositional structure," Ph.D. dissertation, University of Toronto, Toronto, ON, Canada, 1994.
[21] D. Kleyko, "Vector symbolic architectures and their applications: Computing with random vectors in a hyperdimensional space," Ph.D. dissertation, Luleå University of Technology, 2018.
[22] I. Barclay et al., "Trustable service discovery for highly dynamic decentralized workflows," *Future Gener. Comput. Syst.*, vol. 134, pp. 236–246, 2022.
[23] P. Kanerva, "Computing with high-dimensional vectors," *IEEE Des. Test*, vol. 36, no. 3, pp. 7–14, Jun. 2019.
[24] D. A. Rachkovskij, "Shift-equivariant similarity-preserving hypervector representations of sequences," 2021, *arXiv:2112.15475*.
[25] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artif. Intell. Rev.*, vol. 55, no. 6, pp. 4523–4555, 2022.
[26] P. Neubert, S. Schubert, and P. Protzel, "An introduction to hyperdimensional computing for robotics," *KI-Künstliche Intelligenz*, vol. 33, pp. 319–330, 2019.
[27] D. Kleyko et al., "Vector symbolic architectures as a computing framework for emerging hardware," in *Proc. IEEE*, vol. 110, no. 10, pp. 1538–1571, Oct. 2022.
[28] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–25, 2019.
[29] V. I. Gritsenko, D. A. Rachkovskij, A. A. Frolov, R. Gayler, D. Kleyko, and E. Osipov, "Neural distributed autoassociative memories: A survey," 2017, *arXiv: 1709.00848*.
[30] E. P. Frady, D. Kleyko, and F. T. Sommer, "Variable binding for sparse distributed representations: Theory and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2191–2204, May 2023.
[31] V. Shirhatti, A. Borthakur, and S. Ray, "Effect of reference scheme on power and phase of the local field potential," *Neural Comput.*, vol. 28, no. 5, pp. 882–913, 2016.
[32] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part II: Applications, cognitive models, and challenges," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–52, 2023.
[33] C. Simpkin, I. Taylor, D. Harborne, G. Bent, A. Preece, and R. K. Ganti, "Efficient orchestration of node-red IoT workflows using a vector symbolic architecture," *Future Gener. Comput. Syst.*, vol. 111, pp. 117–131, 2020.
[34] L. Wang, H. Chen, and W. Qin, "Nbuflow: A dataflow based universal task orchestration and offloading platform for low-cost development of IoT systems with cloud-edge-device collaborative computing," in *Proc. 21st Int. Conf. Algorithms Architectures Parallel Process.*, 2021, pp. 665–681.
[35] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cogn. Comput.*, vol. 1, no. 2, pp. 139–159, 2009.
[36] S. Datta, R. A. Antonio, A. R. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric IoT," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 3, pp. 439–452, Sep. 2019.
[37] Z. Zou, Y. Kim, M. H. Najafi, and M. Imani, "ManiHD: Efficient hyperdimensional learning using manifold trainable encoder," in *Proc. Des. Automat. Test Eur. Conf. Exhib.*, 2021, pp. 850–855.
[38] R. Xie, Z. Wang, F. R. Yu, T. Huang, and Y. Liu, "A novel identity resolution system design based on dual-chord algorithm for industrial Internet of Things," *Sci. China Inf. Sci.*, vol. 64, no. 8, 2021, Art. no. 182301.
[39] Y. Zhang, F. Zhang, H. Li, S. Zhang, and X. Du, "CompressStreamDB: Fine-grained adaptive stream processing without decompression," in *Proc. 39th IEEE Int. Conf. Data Eng.*, Anaheim, CA, USA, 2023, pp. 408–422.

**Haiming Chen** (Member, IEEE) received the BEng and MEng degrees in computer engineering from Tianjin University, China, in 2003 and 2006, respectively, and the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010. He is now an associate professor with the department of computer science, Ningbo University. From 2010 to 2017, he was a research assistant professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He also was a visiting scholar from 2015 for one year with the department of computer science, College of William and Mary, Williamsburg, VA, USA. His research interests include Internet of Things and Edge-Cloud Collaborative Computing Systems. He is a senior member of the CCF.

**Lei Wang** received the MEng degree from the Faculty of Electrical Engineering and Computer Sciences, Ningbo University, in 2023. His research interests include low code development of IoT systems and service discovery in distributed systems.

**Wei Qin** received the MEng degree from the Faculty of Electrical Engineering and Computer Sciences, Ningbo University, in 2023. His research interests include intelligent edge computing in Internet of Things.

**Xinyan Zhou** received the BEng and PhD degrees in electrical engineering from Zhejiang University, Hangzhou, China, in 2014 and 2019, respectively. She is currently an associate professor with the Faculty of Electrical Engineering and Computer Science of Ningbo University. Her research interests include IoT security and wireless communication protocol design.

**Li Cui** (Member, IEEE) received the undergraduation degree from Tsinghua University, China, in 1985, and the MSc degree from the Institute of Semiconductors, Chinese Academy of Sciences, in 1988, and the PhD degree from the University of Glasgow, U.K., in 1999. She was a visiting scholar in the Department of Chemistry with the University of Toronto, Canada, in 1994. In 1995, she worked as a Royal Society Fellow and, from 1996 to 2003, a research associate in the Department of Electronics and Electrical Engineering with the University of Glasgow. She received "The 100 Talents Program of CAS" award in 2004 from CAS and is currently a professor with the Institute of Computing Technology, Chinese Academy of Sciences. Her current research is focused on sensor technology, wireless sensor networks, and Internet of Things. She is an IET Fellow.