HyperDetect: A Real-Time Hyperdimensional Solution for Intrusion Detection in IoT Networks

Junyao Wang, *Graduate Student Member, IEEE*, Haocheng Xu, *Graduate Student Member, IEEE*, Yonatan Gizachew Achamyeleh, Sitao Huang, *Member, IEEE*, and Mohammad Abdullah Al Faruque, *Senior Member, IEEE*

***Abstract*—Network-based security has emerged as an increasingly critical challenge in the domain of the Internet of Things (IoT). A number of network intrusion detection systems (NIDS), typically relying on sophisticated machine learning (ML) algorithms, have been proposed to monitor network traffic and detect malicious activity. However, these NIDS designs require extensive memory and computational power, exceeding the capability of today's IoT devices, and often fail to provide timely detection of network attacks. To tackle this issue, we propose HyperDetect, the first attempt at NIDS modeling that leverages the highly efficient and parallel operations of brain-inspired hyperdimensional computing (HDC). Our innovative model updating method effectively mitigates model saturation and significantly reduces the number of retraining iterations needed to reach convergence. Additionally, we employ a novel dynamic encoding technique to regenerate insignificant dimensions, considerably lowering the dimensionalities required to achieve high-quality performance and further accelerating the learning process. HyperDetect delivers on average 5.02× faster training and 31.83× faster inference compared to state-of-the-art (SOTA) learning approaches on a wide range of network intrusion classification tasks. We also extensively evaluate HyperDetect on embedded hardware to demonstrate its low-latency and resource-efficient characteristics.**

***Index Terms*—Bio-inspired learning, hyperdimensional computing (HDC), Internet of Things (IoT), network intrusion detection.**

I. Introduction

THE Internet of Things (IoT) has recently become an emerging trend for its extraordinary potential to connect various heterogeneous smart sensors and devices. However, notorious IoT attacks, such as Stuxnet [1], have raised both social and industrial concerns regarding network-based security issues. In particular, due to the interconnected nature of these devices, compromising a single component or communication channel in Internet of Things (IoT)-based systems can potentially paralyze the entire network [2], [3]. Traditional anti-virus software and firewalls perform less effectively against the evolving landscape of cyber-threats; network intrusion detection systems (NIDS) have become one of the most widely deployed tools to protect information infrastructures in the past two decades [4], [5]. As demonstrated in Fig. 1, when traditional firewalls fail to intercept intruders, NIDS step in to provide real-time detection and alerts to mitigate the attacks [6]. However, existing NIDS models can be extremely challenging to deploy given the resource limitations and potential instabilities of IoT systems, more resource-efficient and hardware-friendly network intrusion detection solutions are of absolute necessity [7], [8].

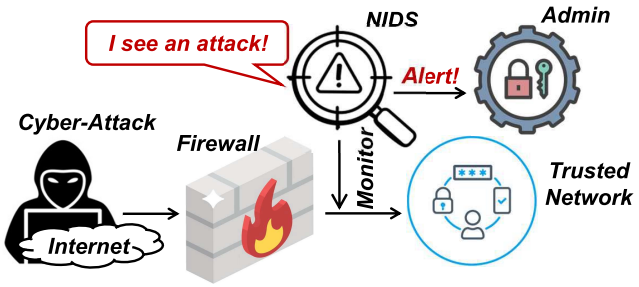


Fig. 1. Model of NIDS.

Popular NIDS designs rely heavily on machine learning (ML) models to achieve high-quality performance [9], [10], [11]. However, NIDS based on traditional ML algorithms, e.g., support vector machines (SVMs), requires considerable feature engineering and fine-tuning to provide adequate detection accuracy, and demands consistent maintenance with up-to-date training data sets to identify constantly evolving cyber-threats [12], [13], [14], [15]. Although NIDS models developed upon deep learning (DL) generally perform better by learning from raw data, their excellent learning quality often comes at the expense of high computational and memory requirements, involving millions of parameters iteratively refined over multiple time periods [9], [16]. These resource-intensive NIDS designs can be impractical for IoT systems. While today's common approach is to send data from edge to the centralized location in the cloud to complete sophisticated learning and training tasks, it can potentially

Manuscript received 6 September 2023; revised 6 October 2023 and 17 November 2023; accepted 16 December 2023. Date of publication 20 December 2023; date of current version 9 April 2024. *(Corresponding author: Junyao Wang.)*

Junyao Wang is with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: junyaow4@uci.edu).

Haocheng Xu, Yonatan Gizachew Achamyeleh, and Sitao Huang are with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: haochx5@uci.edu; yachamye@uci.edu; sitaoh@uci.edu).

Mohammad Abdullah Al Faruque is with the Department of Computer Science and the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: alfaruqu@uci.edu).

Digital Object Identifier 10.1109/JIOT.2023.3345279

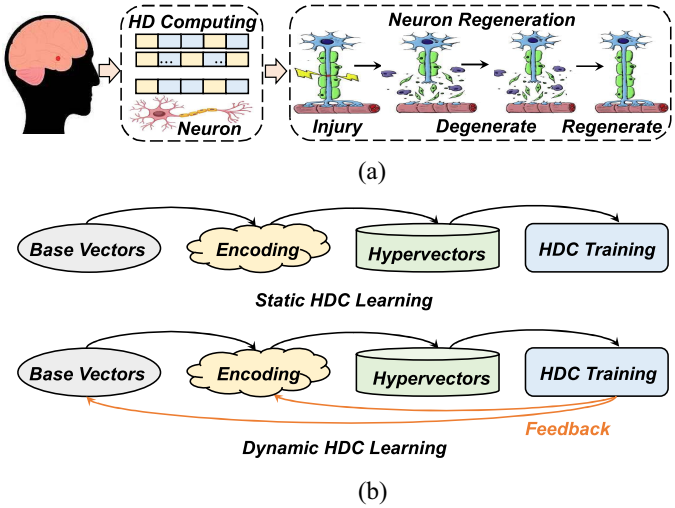


Fig. 2. (a) Motivation of our proposed HDC learning framework comes from the dynamic neuron regeneration of human brains. (b) Comparing static and our proposed dynamic HDC learning frameworks.

cause drastic efficiency loss, incur serious scalability issues, and even raise new security concerns [17], [18]. Edge-based computing, distributing learning tasks onto the IoT hierarchy and bringing computations close to data sources, is hence considered a more promising solution. Given the increasingly massive amount of network traffic and the real-time requirement of network intrusion detection, there is an imperative need for a highly efficient and lightweight NIDS design to address security issues in IoT systems.

In contrast to popular ML methodologies, hyperdimensional computing (HDC) is considered a promising learning algorithm for resource-constrained IoT platforms for its 1) high computational efficiency ensuring real-time learning; 2) strong robustness against noise—a key strength for IoT systems; and 3) lightweight hardware implementations allowing efficient execution on edge [19], [20], [21], [22]. As demonstrated in Fig. 2(a), HDC originates from the neuroscience observation that the cerebellum cortex in human brains effortlessly and efficiently processes memory, perception, and cognition tasks with neural activities in high-dimensional space. Closely mimicking information representations in human brains, HDC *encodes* low-dimensional inputs to *hypervectors* consisting of thousands of elements to perform learning tasks with highly parallel and well-trackable operations. Recent research has shown that HDC is capable of achieving high-quality results with notably faster convergence than state-of-the-art (SOTA) learning approaches [21]. Additionally, utilizing encoded data points on high-dimensional space, HDC can potentially bring unique advantages in distinguishing various sophisticated attacks, especially nowadays when attacks are often disguised with similar patterns of normal network traffic.

Nevertheless, existing HDCs have two major drawbacks: 1) they do not consider momentum in the model, easily resulting in model saturation and requiring hundreds of iterations to converge and 2) they typically use pregenerated encoding modules that are never updated during the entire training phase and thus require extremely high dimensionalities to achieve acceptable accuracy. These drawbacks not only severely lower the learning efficiency by involving large amounts of unnecessary computations but also compromise the system efficiency with increased data size and communication cost [23]. This can be particularly destructive for performing today's network intrusion detection tasks, which often require efficiently monitoring and analyzing billions of network traffic instances. We observe one major cause is that the encoding module of existing HDCs is incapable of utilizing and adapting to information learned during training, and hence often fails to find a good representation of the data with lower dimensionalities. In contrast, as shown in Fig. 2(a), neurons in human brains dynamically change and regenerate all the time and provide more useful functionality when accessing new information. Specifically, every day, approximately 85 000 neurons die, i.e., 31 million in a year, and a similar number of neurons are generated simultaneously to provide more useful functionality to the brain [24], [25], [26]. While the goal of HDC is to utilize the high dimensionality of randomly generated vectors to represent information as a pattern of neural activity, existing HDCs can hardly support a similar behavior.

To address this issue, we propose HyperDetect, the first network intrusion classification model leveraging the highly parallel operations provided by brain-inspired HDC. We introduce a novel model updating method explicitly considering the momentum at each data point. Additionally, as shown in Fig. 2(b), unlike existing HDCs performing encoding and training sequentially in a one-way fashion, HyperDetect works bidirectionally, enabling base vectors and encoding modules with adaptivity to the information learned from each training iteration. HyperDetect thereby provides an optimized model that achieves effective intrusion detection with notably fewer training iterations and lower dimensionalities, significantly accelerating both training and inference by eliminating unnecessary computations. The main contributions of this article are listed as follows.

1) To the best of our knowledge, HyperDetect is the first NIDS model leveraging the highly efficient and parallel operations of HDC to deliver real-time attack detection. HyperDetect provides on average a 5.02× faster training and a 3.34× faster inference compared to SOTA DNNs on a wide range of network intrusion classification tasks.
2) We propose a novel model updating method with explicit consideration of the momentum at each data point. Our learning algorithm effectively mitigates model saturation and significantly reduces the number of retraining iterations required to reach convergence.
3) We employ an innovative dimension regeneration technique and optimize it with highly parallel matrix-wise operations. Compared to SOTA HDC with static encoders, HyperDetect reduces the required dimensionality by 16.0× and demonstrates on average a 31.83× speedup in inference.
4) We conduct a thorough design space search utilizing a genetic algorithm (GA) to understand model performance in various hyperparameter settings. Our results demonstrate the great potential of implementing

both training and inference of HyperDetect on edge.

5) We propose hardware-aware optimizations for the implementation of HyperDetect, and evaluate it across multiple embedded devices, including Raspberry Pi, NVIDIA Jetson Nano, and FPGA. HyperDetect provides considerably lower inference latency than other ML-based approaches, with speedups from 1.16× to 4 715.6×, ensuring timely support for attack detection on resource-constrained IoT devices.

II. Background and Related Work

A. Threat Model

Here, we introduce potential network-based vulnerabilities and the threat model in IoT systems by outlining threat agents, attack vectors, system vulnerabilities, and potential impacts.

1) Threat Agents: The primary perpetrators of network-based attacks are external attackers and insider threats. External attackers, i.e., malicious actors outside the network, actively seek out vulnerabilities to exploit and gain unauthorized access. In contrast, insider threats originate from the misuse of authorized access by individuals within the network who possess legitimate access privileges, either intentionally or inadvertently. Both of these perpetrators pose significant risks to network security, requiring real-time network intrusion detection and comprehensive security measures.

2) Attack Vectors: The resource-constrained nature of IoT devices substantially broadens the vulnerability landscape for network-based attacks. Noteworthy attack vectors include port scanning, packet sniffing, IP spoofing, man-in-the-middle attacks, Denial of Service (DoS), Distributed DoS (DDoS), and malware propagation. Considering the limited processing power and memory of IoT devices, each of these tactics presents a unique challenge for network intrusion detection. Therefore, lightweight and adaptive intrusion detection mechanisms become an absolute necessity.

3) System Vulnerabilities: The main vulnerabilities in IoT systems stem from weak authentication mechanisms, unpatched or outdated systems, and inadequate or misconfigured network monitoring. Resource-constrained IoT devices are particularly susceptible to attacks due to insufficient monitoring, which often fails to identify malicious activity. This results in a broader spectrum of exploitation opportunities for attackers and can potentially cause serious damage.

4) Potential Impacts: A successful network-based attack can potentially cause severe consequences that impact various aspects of IoT systems, including 1) data breaches, i.e., theft or exposure of sensitive data; 2) disrupted operation of IoT devices or even the entire network; and 3) unauthorized access to privilege areas of the network gained by attackers. Additionally, malware propagation can potentially occur, wherein malware within an IoT system spreads throughout the entire network, further compromising the system's security.

B. Application of ML in NIDS

1) Popular NIDS Implementation: NIDS are designed to monitor large amounts of network traffic and identify malicious activity. Once an abnormal behavior is detected, NIDS dispatches real-time alerts to administrators to mitigate the attack. Prevailing NIDS implementations can be classified into two major categories: 1) signature-based and 2) anomaly-based [10]. A signature-based NIDS protocol maintains a collection of signatures, each of which characterizes the profile of a known security threat, and appropriate action is taken when a traffic instance matches a signature [27], [28]. In contrast, an anomaly-based NIDS design monitors network traffic and compares it to an established baseline of normal traffic profile, sending alerts to the administrator when a received traffic instance is significantly different from the baseline. However, it can often be highly subjective to decide what can be considered as *normal* [29]. Thus, our work, HyperDetect, focuses on the signature-based NIDS setting and is evaluated by signature-based NIDS data sets.

2) Data Set and Learning Approaches for NIDS: Numerous NIDS models leveraging the excellent performance of ML models have been proposed in the past decade. A number of cyber-security data sets consisting of real-life and automatically generated network traffic have also been established as effective benchmarks for comparing different NIDS designs. For earlier well-known data sets, such as NSL-KDD [30] and UNSW-NB15 [31], models based on SVMs [13], [14], [15] have consistently achieved excellent performance. However, SVM-based learning models require substantially long time periods for both training and inference when confronting large amounts of data samples. Since the release of two more recent data sets CIC-IDS-2017 [32] and CIC-IDS-2018 [33], which contain significantly larger amounts of network traffic and more sophisticated attack patterns, SVMs become less practical and more sophisticated DL models have been actively developed [15], [34], [35]. Nevertheless, these sophisticated DL models require extensive computational and memory resources and are often impractical for resource-constrained embedded devices [16], [36]. In contrast, our HyperDetect utilizes HDC to capture intricate attack patterns with efficient and parallel matrix operations on high-dimensional space, and thereby provide a more resource-efficient NIDS solution for IoT systems.

C. Hyperdimensional Computing

Prior studies have exhibited enormous success in various applications of HDC, such as graph reasoning [37] and language recognition [21]. These works provide comparable accuracy to SOTA learning approaches and require significantly lower energy and computational resources. However, existing HDC learning frameworks do not consider momentum at each data point and use pregenerated static encoders that are never updated during training. Consequently, hundreds of training iterations and extremely high dimensionalities are required to achieve acceptable accuracy [38]. NeuralHD [23], a recently proposed dynamic encoding approach, successfully compressed the required dimensionality by eliminating dimensions with minor impacts on distinguishing patterns. However, its proposed model bundles encoded data samples sequentially in a scalar manner and delivers significantly

lower learning efficiency than SOTA HDCs with static encoders [21], [39]. This can cause serious problems for real-world network intrusion detection where there are often stringent time requirements. In particular, long inference latency can directly cause an exponential increase in the risk of network security posed by intruders. To the best of our knowledge, HyperDetect is the first time that a dynamic HDC framework is being utilized to tackle the critical network intrusion detection tasks on resource-constrained IoT devices. We formulate and fully optimize our model with highly parallel matrix operations, thereby tremendously accelerating both training and inference. We also propose an innovative model updating scheme that explicitly considers momentum at each data point so that the number of retraining iterations can be significantly reduced. This ensures powerful real-time support for attack detection, positioning HyperDetect an ideal NIDS model for IoT systems.

III. Hyperdimensional Classification

HDC, a novel computational framework inspired by cognitive neuroscience, utilizes high-dimensional vectors for information processing. It leverages great amounts of expressive power on high-dimensional space to model associations between data samples. One unique property of the high-dimensional space is the existence of a large number of nearly orthogonal hypervectors. Specifically, in spaces with a large enough dimensionality, two random vectors are almost guaranteed to be within 5 degrees of orthogonal [40]. Mathematically, consider two random hypervectors $\mathcal{H}_1$ and $\mathcal{H}_2$ with dimension $\mathcal{D}$, when $\mathcal{D}$ is large enough, the dot product $\mathcal{H}_1 \cdot \mathcal{H}_2 \approx 0$. This property enables highly efficient and parallel operations, such as similarity calculations, bundlings, and bindings (elaborated in Section III-B). Here, we introduce basic operations and the learning framework of existing HDCs.

A. Encoding

Inspired by information representation of human brains, HDC starts with encoding samples to high-dimensional space via multiplication with a projection matrix consisting of randomly generated high-dimensional base vectors. Each encoded sample is referred to as a *hypervector*, which contains thousands of elements. A hypervector stores all the information across all its elements so that no element is more responsible for storing any piece of information than another. The encoding technique highly depends on the type of the original data, e.g., feature data, text-like data, and time-series data. Nevertheless, a fundamental principle of encoding remains consistent: the distance correlation in original data should be properly preserved during encoding. This ensures that the encoded representation retains the essential relationships and structures in the original data set and thereby prevents information loss. Popular encoding techniques include multiplication of projection matrices [23], fractional binding [41] that preserves real numbered difference, and time-series encoding [19] that captures and preserves spatial and temporal dependencies. Most of these encoding techniques are based on the basic operations introduced in Section III-B.

B. Basic Operations in HDC

1) *Similarity:* Calculation of the distance between the query hypervector and the class hypervector. For real-valued hypervectors, a common measure is cosine similarity. For bipolar hypervectors, it is simplified to the Hamming distance.
2) *Bundling* (+): Element-wise addition of multiple hypervectors, e.g., $\mathcal{H}_{\text{bundle}} = \mathcal{H}_1 + \mathcal{H}_2$, generating a hypervector with the same dimension as inputs. In high-dimensional space, bundling works as a memory operation and provides an easy way to check the existence of a query hypervector in a bundled set. In the previous example, $\delta(\mathcal{H}_{\text{bundle}}, \mathcal{H}_1) \gg 0$ while $\delta(\mathcal{H}_{\text{bundle}}, \mathcal{H}_3) \approx 0$ ($\mathcal{H}_3 \neq \mathcal{H}_1, \mathcal{H}_2$). Bundling models how human brains *memorize* input information.
3) *Binding* (*): Element-wise multiplication associating two hypervectors to create another hypervector $\mathcal{H}_{\text{bind}}$ that is nearly orthogonal to both $\mathcal{H}_1$ and $\mathcal{H}_2$, i.e., $\mathcal{H}_{\text{bind}} = \mathcal{H}_1 * \mathcal{H}_2$, where $\delta(\mathcal{H}_{\text{bind}}, \mathcal{H}_1) \approx 0$ and $\delta(\mathcal{H}_{\text{bind}}, \mathcal{H}_2) \approx 0$. Due to reversibility, i.e., $\mathcal{H}_{\text{bind}} * \mathcal{H}_1 = \mathcal{H}_2$, information from both hypervectors can be preserved. Binding models how human brains *connect* input information, i.e., associating the information of multiple objects into a single hypervector.

C. HDC Learning

1) Training: After generating each encoded hypervector $\mathcal{H}^l$ for class l, the training module calculates the class hypervector $\mathcal{C}_l$ by bundling all $\mathcal{H}^l$s to find the universal property within a class. Mathematically, $\mathcal{C}_l = \sum_{j=1}^{\mathcal{J}} \mathcal{H}^l$, where $\mathcal{J}$ denote the number of inputs with label l. A trained HDC model consists of k class hypervectors each with dimensions $\mathcal{D}$, where k denotes the number of classes. During retraining, we discard the mispredicted queries from the corresponding mispredicted classes and bundle them into their correct classes. For an encoded data sample $\mathcal{H}$, if is mispredicted as label l' while its true label l, we update the class hypervectors $\mathcal{C}_l$ and $\mathcal{C}_{l'}$ as

$$\begin{aligned} \mathcal{C}_l &\leftarrow \mathcal{C}_l + \eta \cdot [1 - \delta(\mathcal{H}, \mathcal{C}_l)] \times \mathcal{H} \\ \mathcal{C}_{l'} &\leftarrow \mathcal{C}_{l'} - \eta \cdot [1 - \delta(\mathcal{H}, \mathcal{C}_{l'})] \times \mathcal{H} \end{aligned} \tag{1}$$

where η is a learning rate, and $\delta(\mathcal{H}, \cdot)$ denotes the similarity score between $\mathcal{H}$ and the corresponding class hypervector. However, HDC retraining can be computationally expensive as it requires both associative search and model update with high dimensionalities [39]. Therefore, in this article, we aim to provide effective network intrusion detection with lower dimensionalities and faster convergence.

2) Inference: The HDC inference consists of two major steps: 1) encoding inference samples into hyperdimensional space as a query hypervector $\mathcal{Q}$ with the same encoding technique in training and 2) calculating the similarity score between $\mathcal{Q}$ and each class hypervector. The inference sample $\mathcal{Q}$ is then classified to the class to which it achieves the highest cosine similarity score. Hamming distance is often used for binary representation, while the cosine distance is a common measure for hypervectors with high precision.

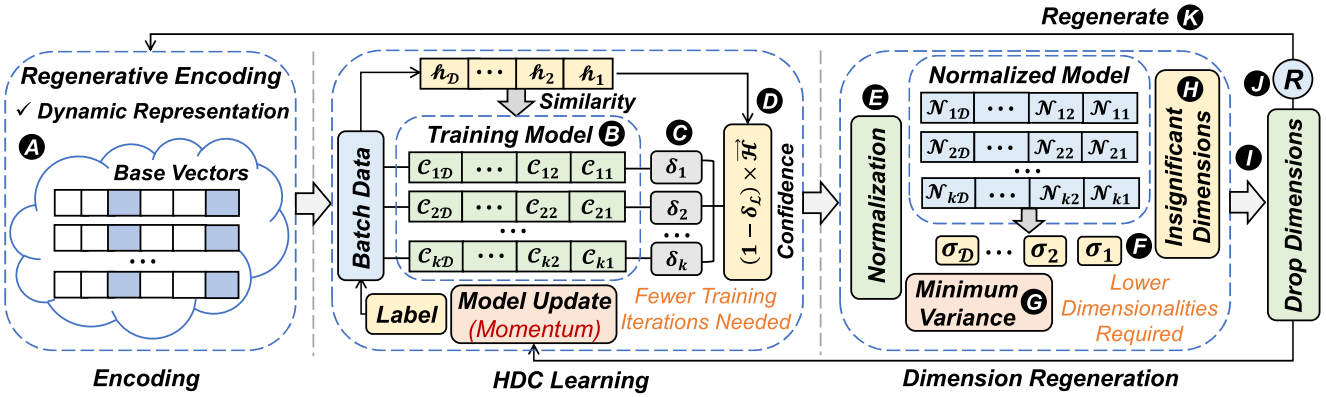


Fig. 3. Overview of the workflow of our proposed HyperDetect. HyperDetect starts with encoding training samples into a high-dimensional space. We then construct HDC models with our innovative model updating technique that explicitly considers the momentum at each data point. Following this, we identify and regenerate dimensions that have minimal impact on the classification task, eventually formulating a highly efficient HDC model.

IV. Methodology

We propose HyperDetect, a resource-efficient HDC learning framework consisting of two innovative steps: 1) *HDC learning* and 2) *dimension regeneration*. In *HDC learning*, we propose a novel adaptive model updating method that 1) eliminates model saturation by scaling a proper weight to each data point according to how much new information is added to the class hypervectors and 2) accelerate model convergence by calculating the momentum at each data point. It effectively lowers the number of retraining iterations required to reach convergence. In *dimension regeneration*, we identify and regenerate insignificant dimensions in the model, and thereby reduce the required dimensionality to achieve adequate accuracy and further accelerate the learning process.

A. Encoding

As shown in Fig. 3, HyperDetect starts with encoding (A) training samples onto high-dimensional space. For network intrusion classification, considering the nonlinear relationship between features, we utilize an encoding technique inspired by the radial basis function (RBF) [42]. Mathematically, for a feature vector $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}(f_i \in \mathbb{R}^n)$ with n features, we generate the corresponding hypervector $\mathcal{H} = \{h_1, h_2, \ldots, h_\mathcal{D}\}(0 \leq h_i \leq 1, h_i \in \mathbb{R})$ with $\mathcal{D}$ dimensions as

$$h_i = \cos(\mathcal{B}_i \cdot \mathcal{F} + c) \times \sin(\mathcal{B}_i \cdot \mathcal{F}) \tag{2}$$

where $\mathcal{B}_i = \{b_1, b_2, \ldots, b_n\}$ is a randomly generated base vector such that $b_i \sim Gaussian(\mu = 0, \sigma = 1)$ and $c \sim Uniform[0, 2\pi]$. The encoding module maps this vector into a high-dimensional vector with length $\mathcal{D}$, where $\mathcal{D} \gg n$. The random vectors $\mathcal{B}_i = \{b_1, b_2, \ldots, b_n\}$ can be generated once offline and then can be used for the rest of the classification tasks $(\mathcal{B}_i) \in \mathbb{R}^m$. After this step, each element h_i represents an element of the hypervector $\mathcal{H}$.

B. HDC Training With Model Momentum

After encoding training samples onto the high-dimensional space, we exploit a highly efficient and parallel HDC learning algorithm (B), bundling each encoded data sample by scaling a proper weight to each of them depending on how much new information is added to class hypervectors. For a new encoded training sample $\mathcal{H}$, we update the model based on its cosine similarities with all class hypervectors (C), i.e.,

$$\delta(\mathcal{H}, \mathcal{C}_l) = \frac{\mathcal{H} \cdot \mathcal{C}_l}{\|\mathcal{H}\| \cdot \|\mathcal{C}_l\|} = \frac{\mathcal{H}}{\|\mathcal{H}\|} \cdot \frac{\mathcal{C}_l}{\|\mathcal{C}_l\|} \propto \mathcal{H} \cdot \mathcal{N}_l \tag{3}$$

where $\mathcal{H} \cdot \mathcal{C}_l$ is the dot product between $\mathcal{H}$ and a class hypervector $\mathcal{C}_l$, and $\mathcal{N}_l$ represents the normalized class hypervector, i.e., $(\mathcal{C}_l/\|\mathcal{C}_l\|)$. Here, $\|\mathcal{H}\|$ is a constant factor when comparing a query with all classes and thus can be eliminated. The calculation of cosine similarity can hence be simplified to a dot product operation. However, the model updating technique of existing HDCs, as shown in (1), only considers the current learning rate and the label at that moment. Specifically, it does not take the last few steps into account while traversing the class hypervectors for the training process. This can potentially lead to minor or no updates that stagnate the learning process, or large updates that make the learning process unable to relax. To address this issue, our novel updating technique explicitly considers the momentum from the previous update. In particular, if $\mathcal{H}$ has the highest cosine similarity with class $\mathcal{L}_i$ while it actually has label $\mathcal{L}_j$, the model updates (D) as

$$\begin{aligned} \mathcal{C}_j^t &\leftarrow \mathcal{C}_j^t + \eta \cdot (1 - \delta_j^t) \times \mathcal{H} + \epsilon \cdot \mathcal{C}_j^{t-1} \\ \mathcal{C}_i^t &\leftarrow \mathcal{C}_i^t - \eta \cdot (1 - \delta_i^t) \times \mathcal{H} + \epsilon \cdot \mathcal{C}_i^{t-1} \end{aligned} \tag{4}$$

where η denotes a learning rate, t and $t-1$ denotes the current learning trail and the previous training trail, respectively. A large δ_l indicates the input data point is marginally mismatched or already exists in the model, and the model is updated by adding a very small portion of the encoded query $(1 - \delta(\mathcal{H}, \cdot) \approx 0)$. In contrast, a small $\delta(\mathcal{H}, \cdot)$, indicating a noticeably new pattern that is uncommon or does not already exist in the model, updates the model with a large factor $(1 - \delta(\mathcal{H}, \cdot) \approx 1)$. To avoid over-fitting, We do not update the model when $\mathcal{H}$ has the highest cosine similarity with its actual label. Our learning algorithm provides a higher chance for noncommon patterns to be properly included in the model, thereby effectively reducing computationally expensive retraining iterations required to achieve reasonable accuracy. Additionally, including the momentum can effectively prevent

us from abrupt changes and falling in the wrong direction; a large difference between $\mathcal{C}_i^t$ and $\mathcal{C}_i^{t-1}$ indicates the current update may cause an oscillation when the optimization landscape is narrow and steep. More precisely, the momentum can be considered as an exponentially weighted moving average of past gradients. Instead of updating the hypervector only based on the current data point, our learning algorithm utilizes an exponentially weighted moving average of the preceding updates that acts as a form of memory for the optimizer, enabling it to achieve faster convergence, reduce the risk of oscillation, retain the direction it was moving in, and persist in that trajectory, even if the current data point suggests a different direction for the update.

C. Dimension Regeneration

1) Drop Insignificant Dimensions: HDC algorithms represent each class with a class hypervector that encodes the patterns of that class. An effective classifier achieves the desired accuracy by a strong capability to distinguish patterns so that, in the inference phase, query vectors can have very differentiated cosine similarities to each class. In contrast, a weak classifier can hardly find distinct patterns for different classes; this makes the classification task hard as the query may have a close similarity value to multiple classes. Similarly, dimensions with similar values over all classes indicate they store common information across classes and therefore play minimal roles in the classification. As demonstrated in Fig. 3, HyperDetect starts with an initial trained model and normalizes each class hypervectors (E). We then calculate the variance of each dimension over all classes (F) to measure the dispersion of that dimension. In particular, dimensions with minimal variances (G) are considered insignificant (H). We then select $\mathcal{R}$ portion of dimensions with the lowest variance to drop (K), depending on a regeneration rate $\mathcal{R}$ (I). In this way, HyperDetect achieves high-quality performance with a significantly compressed dimensionality, and hence greatly improves the training and inference efficiency.

2) Dimension Regeneration: To mitigate the accuracy loss caused by dropping dimensions, instead of leaving these dimensions blank, HyperDetect regenerates (J) them so that the new dimensions can potentially have a higher impact on the classification task and better distinguish different patterns. During regeneration, HyperDetect replaces each of the base vectors (A) of selected dimensions with another randomly generated hypervector from Gaussian Distribution, in the hope that the new hypervector can have a more positive impact on the classification task or provide a better performance.

3) Retraining: HyperDetect then updates the current model by retraining. Instead of starting training from scratch, HyperDetect updates the values of class hypervectors on the dropped dimensions while other dimensions continue learning based on their existing values. The iterative learning and regeneration continue until HyperDetect finds a model where most dimensions are highly contributing to the classification.

D. Hardware Optimizations

Network intrusion detection tasks typically have stringent requirements for both inference latency and model performance. We apply the following hardware-aware optimizations into the implementation of our proposed HyperDetect to maximize both its learning efficiency and model performance:

1) Multithreading: Leveraging the powerful multicore processor, multithreading facilitates the execution of multiple threads simultaneously and enables overlapping computations. In our work, we employ multithreading to effectively enhance the overall throughput of HyperDetect and fully parallelize operations, including random basis generation, basis regeneration, encoding, and vector normalization.

2) Tiled Matrix Multiplication: Several steps in HyperDetect are naturally matrix operations, e.g., encoding and cosine similarity. We employ a memory hierarchy approach, breaking down these matrices into smaller tiles that fit within the cache. This strategy significantly reduces the frequency of data fetching from the main memory, resulting in enhanced efficiency as data can be repeatedly accessed from cached tiles during matrix multiplication.

3) Kernel Fusion: During HyperDetect implementation, we create custom kernels to fuse original kernels, e.g., the fusion of basis regeneration and encoding. This minimizes the frequency of memory read and write operations required for storing intermediate results back to the main memory due to limitations in local buffer size. Additionally, it effectively reduces the overhead associated with kernel invocations.

4) Quantization: Quantization can significantly reduce the memory requirements by decreasing the number of bits required for storing numerical values. Additionally, it accelerates computations thanks to the innate hardware support for quantized numbers, such as INT8, which is notably faster when compared to higher precision floating-point numbers. HyperDetect supports the customization of bitwidth to fully utilize low-bitwidth functional units on specific hardware platforms, further enhancing computational efficiency.

E. Hyperparameter Design Space Exploration

1) Design Space Definition: To understand HyperDetect in different hyperparameter settings, in particular, for different physical dimensionality ($\mathcal{D}$), effective dimensionality ($\mathcal{D}^*$), and regeneration rate ($\mathcal{R}$), we conduct a comprehensive design space exploration (DSE). The effective dimensionality ($\mathcal{D}^*$) is defined as the addition of the physical dimensions ($\mathcal{D}$) of HyperDetect with all the regenerated dimensions throughout the retraining iterations. Mathematically, $\mathcal{D}^* = \mathcal{D} + \mathcal{D} \times \mathcal{R} \times$ *Number of Iterations*, where $\mathcal{R}$ is a regeneration rate. Existing HDC works [23] typically select the physical dimensionality ($\mathcal{D}^*$) and effective dimensionality ($\mathcal{D}$) based on heuristic, which might not be the optimal solution for IoT network intrusion detection tasks with stringent constraints in both accuracy and hardware resources. Therefore, we define a comprehensive multidimensional design space so that we can explore all the possibilities and systematically search for optimal solutions. We define our design space to be a 3-D

TABLE I
TERMINOLOGY IN GA

Term	Description
Gene	The value of one of the dimensions of a design point
Genome	A set of genes that represents a design point
Population	A set of all possible genomes (one generation)
Elite	A set of genomes with high fitness score
Mutation	Randomly choose some genes to be changed
Crossover	Blend parents' genes to reproduce children genomes

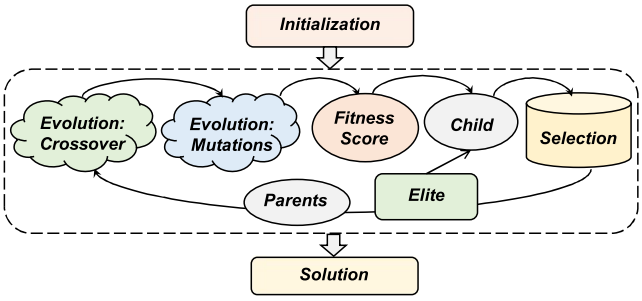


Fig. 4. GA for HyperDetect DSE.

space spanned by all valid values for $\mathcal{D}$, $\mathcal{D}^*$, and $\mathcal{R}$. The size of this space is $\mathcal{O}(10^{11}) = \mathcal{O}(10^3 \times 10^5 \times 10^3)$.

2) Search Algorithm: Due to the vastness of the design space (hundreds of billions of possibilities) and the exponential increase in the evaluation cost for certain hyperparameter variations, it can be impractical to employ a naive exhaustive grid search method to explore the entire design space. Here, we develop a search technique based on GAs, one of the most popular and versatile algorithms for addressing scheduling problems within a discontinuous space. GA can converge to global optima in most hyperparameter optimization problems and achieve comparable performance as deep reinforcement learning [43]. Some common terminologies for GA are demonstrated in Table I.

3) Genetic Algorithm: As shown in Fig. 4, our DSE flow includes *initialization*, two *evolution* operators, i.e., *crossover* and *mutations*, an *evaluation* stage to calculate the *fitness score*, and a *selection* stage to select the good genes. We elaborate on these generic evolution operators as follows.

1) *Initialization:* We select a subset of design points for each generation by randomly initializing a population size $\mathcal{P}$. Specifically, we construct $\mathcal{P}$ using different combinations of physical dimensionality ($\mathcal{D}$), effective dimensionality ($\mathcal{D}^*$), and regeneration rate ($\mathcal{R}$), and each combination is considered a *genome* or an individual.
2) *Evolution Crossover:* We randomly select a pair of genes, i.e., two genomes, from the $\mathcal{P}$ combinations initialized and blend their *genes* by interchanging parameter values.
3) *Evolution Mutations:* We assign a random probability for each pair of genes.
4) *Fitness Score:* After evolution, we use the following fitness function to evaluate the candidates and then select the candidates with the higher fitness score for the next generation:

$$\text{Fitness Score} = \alpha \cdot \mathcal{L} + \beta \cdot \mathcal{A} + \gamma \cdot \mathcal{T} \tag{5}$$

where $\mathcal{L}$ denotes latency, $\mathcal{A}$ denotes accuracy, and $\mathcal{T}$ denotes training time. α, β, γ are hyperparameters that can be tuned. In the experiment, we set $\alpha = 0.5$, $\beta = 0.3$, and $\gamma = 0.2$ to expect a better finding targeting edge devices.

Algorithm 1 Hyperparameter DSE

Input: Sets of valid hyperparameters $\{\mathcal{D}_i\}$, $\{\mathcal{D}_j^*\}$, $\{\mathcal{R}_k\}$
Output: The optimal training time ($\mathcal{T}^*_{\text{train}}$), inference latency ($\mathcal{T}^*_{\text{test}}$), and accuracy ($\mathcal{A}^*$)

for each $(i, j, k) \in \{\mathcal{D}_i\} \times \{\mathcal{D}_j^*\} \times \{\mathcal{R}_k\}$ **do**
  if $i > j$ **then**
    continue ▷ $\mathcal{D}$ should not be greater than $\mathcal{D}^*$
  else
    $\mathcal{T}_{\text{train}}, \mathcal{T}_{\text{test}}, \mathcal{A} \leftarrow \text{HyperDetect}^{(i,j,k)}(data)$
$\mathcal{T}^*_{\text{train}}, \mathcal{T}^*_{\text{test}}, \mathcal{A}^* \leftarrow \text{optimal}(\mathcal{T}_{\text{train}}, \mathcal{T}_{\text{test}}, \mathcal{A})$
return $i, j, k \leftarrow \mathcal{D}, \mathcal{D}^*, \mathcal{R}$

TABLE II
DATA SETS FOR EVALUATING NIDS DESIGNS (n: NUMBER OF FEATURES AND k: NUMBER OF CLASSES)

	n	k	Train Size	Test Size	Release Year
NSL-KDD	33	5	665,319	285,137	2009 [30]
UNSW-15	39	10	180,371	77,302	2015 [31]
CIC-IDS-2017	49	15	1,698,008	727,719	2017 [32]
CIC-IDS-2018	41	11	172,711	74,019	2018 [33]

4) Verification: We employ Algorithm 1 to traverse part of possible combinations of $(\mathcal{D}, \mathcal{D}^*, \mathcal{R})$ to construct a design space that is significantly smaller than the original one and more practical to evaluate. Considering the resource limitations of edge devices, we start by setting a lower bound and upper bound for the physical dimensionality ($\mathcal{D}$) of our HDC model and then emulate from the lowest physical dimensionality toward the upper bound. Additionally, we emulate all the possible effective dimensionality ($\mathcal{D}^*, \mathcal{D}^* \geq \mathcal{D}$) and regeneration rate ($\mathcal{R}$) to fully explore this design space and capture the optima. Algorithm 1 outputs the optimal hyperparameters that deliver the best detection accuracy, training efficiency, and testing efficiency in the smaller space defined by GA. With the trend and results obtained from this search, we can verify the performance of GA (elaborated in Section V-E).

V. EXPERIMENTAL RESULT

We evaluate HyperDetect on widely used network intrusion data sets listed in Table II. We compare HyperDetect with SOTA DNNs [44], SVMs [15], and HDC algorithms [23], [39] in terms of accuracy, training and inference efficiency, and performance on resource-constrained devices. We implement and evaluate HyperDetect on a wide range of platforms, including server CPU, embedded CPU, embedded GPU, and FPGA. We also explore the hyperparameter design space of HyperDetect and identify the optimal hyperparameters.

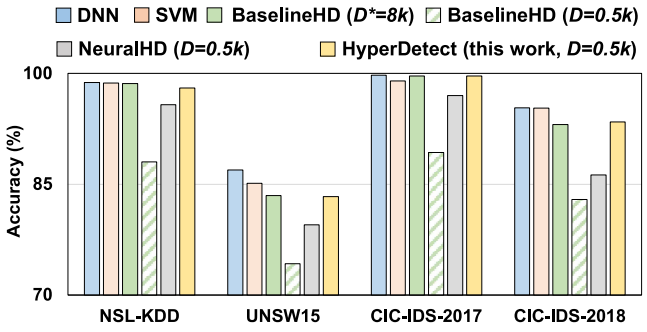


Fig. 5. Comparing accuracy of NIDS classification. HyperDetect demonstrates a comparable classification accuracy to SOTA learning algorithms.

A. Experimental Setup

We evaluate HyperDetect on both server CPU and embedded platforms listed as follows.

1) Server CPU: Intel Xeon Silver 4310 CPU (12-core, 24-thread, 2.10 GHz), 96-GB DDR4 memory, Ubuntu 20.04, Python 3.8.10, PyTorch 1.12.1, and TDP 120 W.

2) Embedded CPU: Raspberry Pi 3 Model 3+ (quad-core ARM A53 @1.4 GHz), 1-GB LPDDR2 memory, Debian 11, Python 3.9.2, PyTorch 1.13.1, and TDP 5 W.

3) Embedded GPU: NVIDIA Jetson Nano (quad-core ARM A57 @1.43 GHz, 128-core Maxwell GPU), 4 GB LPDDR4 memory, Ubuntu 20.04, Python 3.8.10, PyTorch 1.13.0, CUDA 10.2, and TDP 10 W.

4) FPGA: AMD Alveo U50, 8-GB HBM, PCI express (PCIe) gen3 x16, 872K lookup tables (LUTs), 1743K registers, 5952 DSPs, and TDP 75 W.

B. Accuracy

1) HyperDetect Versus Popular Learning Approaches: We compare the accuracy of HyperDetect with the SOTA DNN and SVMs for each data set. Our DNN algorithm is trained with TensorFlow while SVM is trained with the scikit-learn library [45]. We utilize the common practice of grid search to identify the best hyperparameters for each model. As demonstrated in Fig. 5, HyperDetect provides a comparable classification accuracy to these SOTA learning approaches.

2) HyperDetect Versus SOTA HDC: We compare the classification accuracy of HyperDetect with the SOTA HDC algorithm without the capability to regenerate dimensions (BaselineHD) [21] and a recently proposed HDC learning approach with a dynamic encoding technique (NeuralHD) [23]. The results of BaselineHD are reported in two dimensionalities: 1) *physical dimensionality* ($\mathcal{D} = 0.5$k) of NeuralHD and HyperDetect, a compressed dimensionality designed for resource-efficient implementations on IoT devices and 2) *effective dimensionality* ($\mathcal{D}^* = 8$k), defined as the sum of the physical dimensions ($\mathcal{D}$) and all the regenerated dimensions throughout the retraining iterations. As demonstrated in Fig. 5, HyperDetect shows on average a 14.98% higher accuracy than the SOTA HDC ($\mathcal{D} = 0.5$k) and 3.97% higher accuracy than NeuralHD ($\mathcal{D} = 0.5$k). Additionally, HyperDetect exhibits a comparable accuracy to the SOTA HDC ($\mathcal{D}^* = 8$k), indicating HyperDetect is capable of providing high-quality attack detection while using 16.0× lower physical dimensionality.

C. Efficiency

1) Efficiency on Server CPU: For fairness, we compare the training time and inference latency of HyperDetect with the SOTA DNN, SVMs, BaselineHD ($\mathcal{D}^* = 8$k) and NeuralHD ($\mathcal{D} = 0.5$k) since they achieve comparable accuracy as elaborated in Section V-B. As demonstrated in Fig. 6, HyperDetect delivers considerably higher learning efficiency than the SOTA DNN (5.02× faster training, 3.34× faster inference), SVMs (97.79× faster training, 18 959.88× faster inference), BaselineHD (31.83× faster inference), and NeuralHD (234.13× faster training, 225.73 faster inference). In contrast to NeuralHD, where data points are processed by scalar operations sequentially, HyperDetect train data samples with optimized and highly parallel matrix operations. Compared to BaselineHD, though HyperDetect requires more iterations to reach convergence during the training process, each iteration requires significantly less time because of the notably lowered dimensionality; HyperDetect thereby provides comparable training efficiency to BaselineHD. Additionally, such lower dimensionality remarkably accelerates the inference by simplifying the encoding and classification of query vectors.

2) Efficiency on Embedded CPU and GPU: To further understand the performance of HyperDetect on resource-constrained embedded devices, we evaluate the efficiency of HyperDetect, the SOTA DNN, SVMs, and BaselineHD ($\mathcal{D}^* = 8k$) using a Raspberry Pi 3 Model B+ board and an NVIDIA Jetson Nano board. Both platforms have very limited memory and CPU cores (and GPU cores for Jetson Nano). Fig. 7 shows the total inference time for each algorithm processing all 77 302 samples in the UNSW-15 data set. Specifically, HyperDetect outperforms other algorithms on both the Raspberry Pi and the Jetson Nano platforms in terms of inference latency with speedups ranging from 1.16× to 4 715.6×. Notably, BaselineHD ran out of memory and failed to complete the inference task on the Raspberry Pi. We also tested the power consumption as demonstrated in Fig. 7. SVM requires extraordinarily long inference time on both platforms compared to other methods, so we killed the programs; BaselineHD can not even start running inference on Raspberry Pi so the program is also killed. Among algorithms that can be successfully executed on edge devices, HyperDetect demonstrates significantly higher energy efficiency, ranging from 1.5× to 80× more. This evaluation further proves that HyperDetect can run efficiently on energy-constrained platforms.

D. Data Size Scalability

We exhibit the scalability of HyperDetect and SOTA learning algorithms using various training and inference data sizes (percentages of the full data set). As shown in Fig. 8, with the increasing size of the training data set, HyperDetect is capable of maintaining high efficiency for both training and inference with a sublinear growth in execution time. In contrast, the training time of other learning approaches,

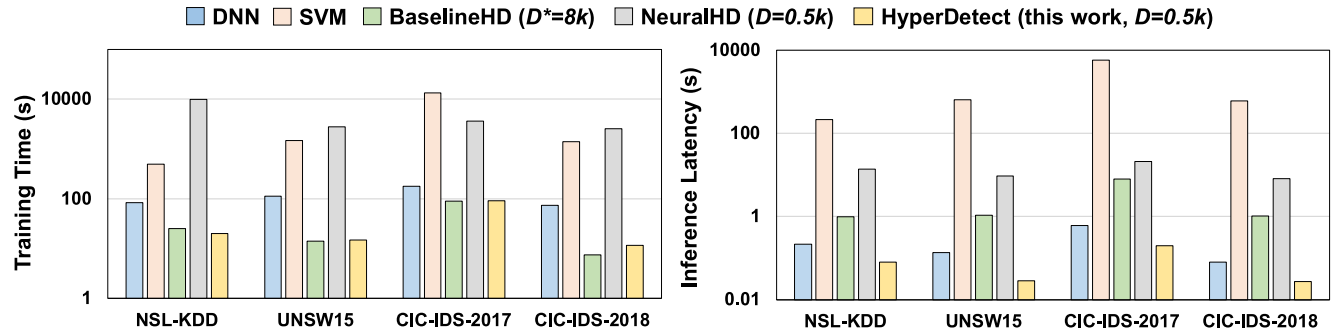


Fig. 6. Comparing training time and inference latency on server CPU. HyperDetect provides 5.02× faster training and 3.34× faster inference than SOTA DNN [44], 97.79× faster training and 18.959.88× faster inference than SVMs [15], 31.82× faster inference than the SOTA HDC [39], and 234.13× faster training and 225.73× faster inference than NeuralHD [23].

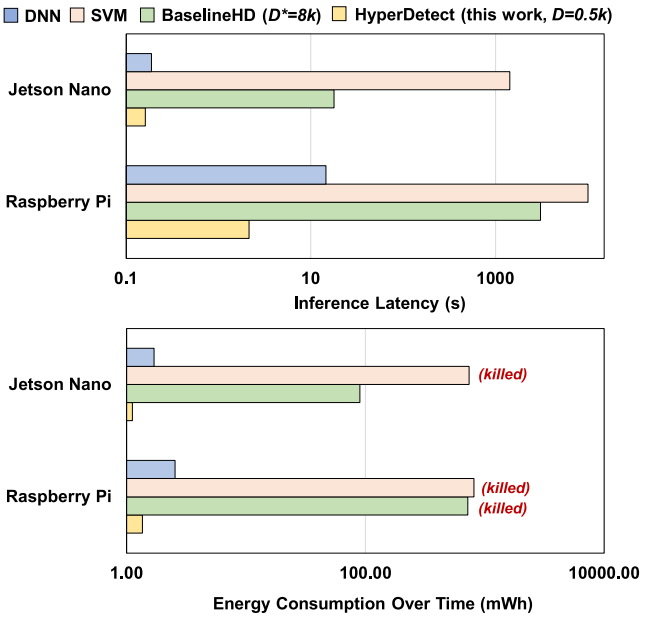


Fig. 7. Inference latency and energy consumption on embedded platforms. HyperDetect provides significantly faster inference and requires notable energy consumption than other learning algorithms.

including the SOTA DNN, SVMs, and BaselineHD ($\mathcal{D}^* = 8$k) increases notably faster than HyperDetect. This indicates that HyperDetect is capable of providing scalable NIDS solutions for both high-performance and resource-constrained computing devices.

E. Hyperparameter Design Space Exploration

1) Setting: The framework of our GA is demonstrated in Fig. 4. We set both the evolution mutation rate and the crossover rate as 0.1, and our GA implementation is adopted from Nevergrad [46]. To verify the finding of GA, we evaluate each design point, with different steps, in the space and analyze how they impact the detection accuracy, training time, and testing time of HyperDetect. We set physical dimensionality ($\mathcal{D}$) to be from 500 to 4000 with 200 increments in each step, effective dimensionality ($\mathcal{D}^*$) to be from 2000 to 8000 with 500 increments in each step, and regeneration rate to be from 10% to 90% with 20% increment in each step, and then follow procedures detailed in Algorithm 1.

2) Result: As demonstrated in Fig. 9(a), the accuracy of utilizing lower physical dimensionality ($\mathcal{D}$) can be compensated by utilizing more iterations of regeneration and larger effective dimensionality ($\mathcal{D}^*$). Even with extremely low physical dimensionality, e.g., ($\mathcal{D} = 0.5$k), a fine-tuned HyperDetect model can achieve comparable performance to using $\mathcal{D} = 8k$, which requires significantly more computational and memory resources. For instance, the training trails in the neighborhood of B delivers comparable performance to training trails near A. In Fig. 9(b), by fixing physical dimension $\mathcal{D} = 500$, we further prove this idea by showing that the model performance can be considerably enhanced with the increase of effective dimensionality. Additionally, as shown in Fig. 10, with proper hyperparameters, HyperDetect can achieve high-quality performance within a short training period (ranging from 0 to 20 s) with timely inference (ranging from 0.06 to 0.08 s), indicating a great potential of implementing both training and inference tasks at the edge. In conclusion, our GA-based HyperDetect provides us with the Pareto-front solution.

F. Training Efficiency on Low-Bitwidth Hardware Accelerator

Quantization can effectively compress HDC models and enhance learning efficiency. We quantize HyperDetect to six different bitwiths to evaluate its efficiency. Lower bitwidth settings will require higher dimensional hypervectors to achieve comparable model performance. For fairness, we tune the number of dimensions for each bitwidth so that all six configurations achieve roughly the same detection accuracy. The configurations are (dimensions in parentheses), 32 bits (1.2k), 16 bits (2.1k), 8 bits (3.6k), 4 bits (5.6k), 2 bits (7.5k), and 1 bit (8.8k). We design FPGA accelerators with custom bitwidths for HyperDetect using C++ high-level synthesis (HLS) tool, AMD-Xilinx Vitis HLS [47]. We use `ap_fixed` types in Vitis HLS to implement fixed point types with custom bitwidths. The synthesized design has been tested on a system with one Intel i9-12900 host CPU and one Xilinx Alveo U50 board, which is connected to the host system via PCIe. The FPGA-CPU communication channel is generated by AMD-Xilinx

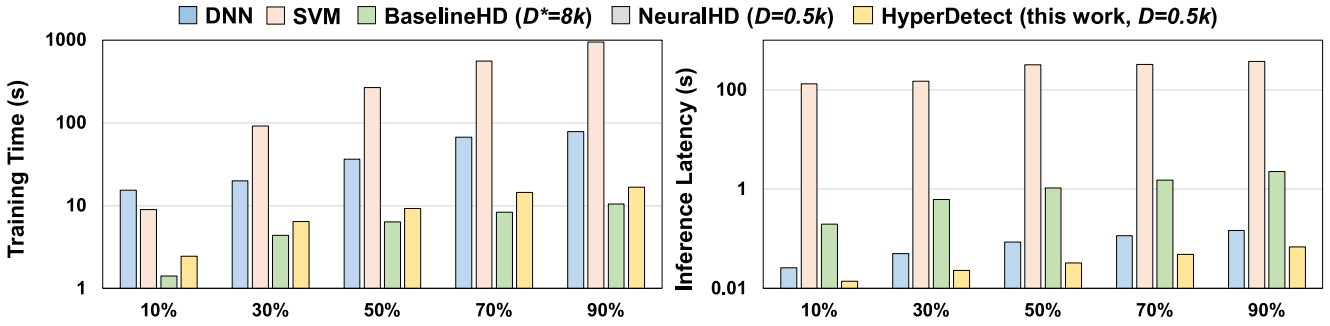


Fig. 8. Comparing training time and inference latency using different sizes of data. HyperDetect maintains significantly higher efficiency for both training and inference than SOTA DNN, SVM, BaselineHD, and NeuralHD, regardless of data sizes.

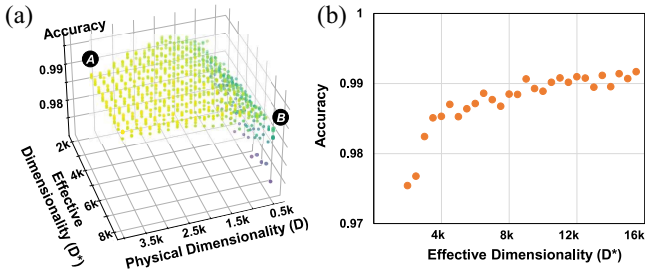


Fig. 9. (a) Relation between physical dimensionality ($\mathcal{D}$), effective dimensionality ($\mathcal{D}^*$), and model performance. (b) Impact of effective dimensionality ($\mathcal{D}^*$) on accuracy with physical dimensionality ($\mathcal{D}$) set as 0.5k. Both (a) and (b) show that, even with very low physical dimensionality (e.g., 0.5k), a fine-tuned model can provide comparable accuracy to models using much higher dimensionalities.

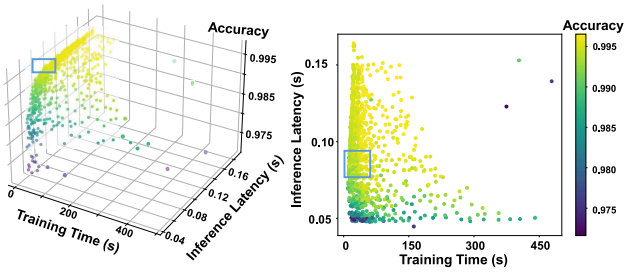


Fig. 10. Relation between training time, inference latency, and model performance. As shown in the blocked areas, high-quality results can be achieved with a short training period and timely inference.

Vitis tool and managed by Xilinx RunTime (XRT). Table III shows the resource utilization on the Alveo U50 FPGA, in terms of LUT, flip-flop (FF), block RAM (BRAM), and DSP. We evaluate the training efficiency of HyperDetect on desktop CPU and this custom FPGA accelerator under various bitwidths, for all four data sets. Fig. 11 shows the evaluation results, including training time breakdown (stacked bars) and speedups from FPGA accelerator over CPU (curve). All FPGA accelerators are running at 200 MHz. HyperDetect on CPU achieves higher efficiency with low dimensionality and high bitwidth because of its high frequency and powerful arithmetic logic unit (ALU). FPGA shows excellent efficiency improvement below 8 bits compared to CPU thanks to FPGA's fine-grained parallelism. Furthermore, this FPGA implementation demonstrates excellent energy efficiency. On the Xilinx Alveo U50 FPGA board, the power consumption

TABLE III
ALVEO U50 FPGA RESOURCE UTILIZATION

bits	$\mathcal{D}$	LUT	FF	BRAM	DSP
32	1.2k	74.09%	56.16%	4.70%	27.25%
16	2.1k	86.63%	59.22%	4.70%	10.04%
8	3.6k	98.35%	58.88%	4.70%	11.55%
4	5.6k	93.04%	24.23%	4.70%	11.55%
2	7.5k	90.08%	21.93%	4.70%	11.55%
1	8.8k	95.02%	23.23%	4.70%	11.55%

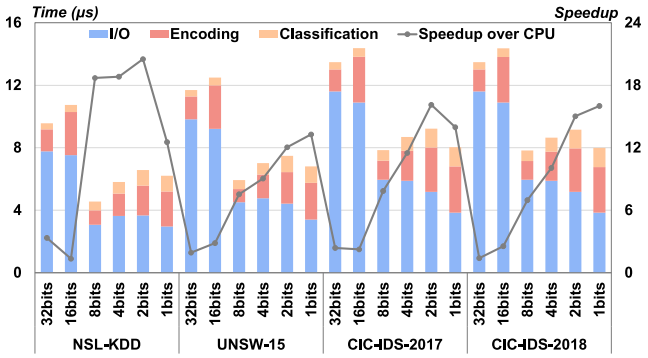


Fig. 11. Efficiency of HyperDetect on low-bitwidth FPGA accelerator.

of the HyperDetect accelerator is less than 20 W under 200-MHz frequency.

G. Robustness Against Errors and Noises

Given the typical harsh deployment environments for IoT systems, we consider the errors and noises caused by potential intrinsic and extrinsic threats and evaluate the robustness of HyperDetect against these errors and noises. We will show that one key advantage of HyperDetect is its high robustness against noise and failure. We evaluate the robustness of HyperDetect and the SOTA DNN-based NIDS model [44] by comparing their average quality loss under different percentages of hardware errors in Fig. 12. The error rate refers to the percentage of random bit flips on memory storing DNN and HyperDetect models. For fairness, all DNN weights are quantized to their effective 8-bit representation. In DNN, random bit flip results in significant quality loss as corruptions on most significant bits can cause major weight changes. In contrast, HyperDetect provides significantly higher robustness against noise due to its redundant and holographic distribution. Specifically, in HyperDetect, every hypervector

Hardware Error			1.0%	2.0%	5.0%	10.0%	15.0%
DNN			3.9%	10.7%	17.8%	32.1%	41.2%
HyperDetect	1-bit	1k	1.1%	1.7%	3.6%	5.4%	7.2%
		2k	0.7%	1.3%	2.8%	4.2%	6.4%
		3k	0.4%	0.7%	1.3%	3.7%	5.9%
		4k	0.0%	0.0%	1.0%	3.1%	4.1%
	2-bits	1k	1.9%	2.3%	4.5%	7.9%	10.4%
		2k	1.2%	1.7%	3.7%	6.8%	9.9%
		3k	0.5%	1.1%	2.5%	5.9%	8.7%
		4k	0.0%	0.5%	1.6%	4.8%	8.0%
	4-bits	1k	2.3%	4.7%	8.4%	13.1%	17.3%
		2k	1.6%	3.2%	6.9%	12.7%	15.9%
		3k	0.9%	2.1%	4.7%	10.2%	13.7%
		4k	0.2%	1.0%	2.9%	7.4%	11.7%
	8-bits	1k	3.6%	7.9%	13.7%	18.3%	22.9%
		2k	2.7%	6.1%	10.8%	15.7%	20.1%
		3k	1.9%	4.9%	8.1%	14.1%	19.8%
		4k	1.4%	3.6%	5.1%	12.8%	17.6%

Fig. 12. Comparing the robustness of HyperDetect and SOTA DNN-based NIDS model against hardware errors.

consists of randomly generated and holographic i.i.d. elements. Each hypervector stores information across all its components so that no component is more responsible for storing any more information than another; therefore, failure on partial data will not result in the loss of entire information. HyperDetect demonstrates the maximum robustness using hypervectors with 4k dimensions in 1-bit precision, that is on average 12.90× higher than the robustness of the DNN. An increase in precision will lower the robustness of HyperDetect since random flips on more significant bits will introduce more loss of accuracy. For instance, for 10% bit flips in hardware, HyperDetect using 1-bit precision and 4k dimensions provides 10.35× and 4.13× higher robustness than the DNN and HyperDetect using 8 bits with the same dimensionality, respectively. Additionally, higher dimensionality improves the robustness of HyperDetect to noise due to its redundant and holographic information distribution. For example, for 10% hardware error, HyperDetect using 4k dimensions and 8-bit precision achieves 1.43× higher robustness than HyperDetect using 1k dimensions with the same bitwidth.

VI. Limitations and Future Works

A. Limitations

In contrast to popular ML and DL learning algorithms, our proposed HyperDetect leverages encoded data on high-dimensional space to provide a real-time and more resource-efficient solution for intrusion detection tasks in IoT Networks. Our proposed HyperDetect also significantly outperforms other HDC learning frameworks in terms of latency and energy consumption. However, the detection accuracy of HyperDetect is slightly lower than DNNs due to the inherent nature of HDC.

1) We utilize the established HDC encoding method to map low-dimensional feature data into a high-dimensional space. Nevertheless, this encoding process may potentially cause information loss and lacks clarity in terms of explainability.
2) The dimension regeneration step involves randomness. While a newly generated vector may potentially contribute positively to classification tasks, such outcomes are not guaranteed and may ultimately limit the model performance that can be achieved.

B. Future Works

For future research, we intend to delve into alternative encoding and dimension regeneration schemes to further enhance our model performance. Additionally, the evaluation of HyperDetect on data-center scale platforms has not yet been conducted. We anticipate that system-level optimizations will be necessary to effectively deploy HDC applications on a larger scale. Currently, the training process of HyperDetect is performed on the host server, but we are actively exploring the possibility of conducting training in an edge environment as well. Finally, as HDC has demonstrated its effectiveness in network intrusion detection tasks, we believe it can also be applied to other security domains, e.g., malware detection.

VII. Conclusion

We propose HyperDetect, a novel HDC framework ensuring resource-efficient and real-time intrusion detection in IoT Networks. HyperDetect dynamically identifies and regenerates dimensions with less impact on classification tasks, and hence effectively reduces the required dimensionality to achieve adequate model performance. Our evaluations on a wide range of network intrusion detection tasks show that HyperDetect delivers significantly higher learning efficiency than existing HDCs and SOTA DNNs. Additionally, HyperDetect outperforms other ML methods on embedded CPU and GPU devices in terms of both inference time and energy efficiency.

References

[1] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, May 2011.

[2] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An in-depth analysis of IoT security requirements, challenges, and their countermeasures via software-defined security," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10250–10276, Oct. 2020.

[3] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for IoT security based on learning techniques," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2671–2701, 3rd Quart., 2019.

[4] A. N. Jahromi, H. Karimipour, A. Dehghantanha, and K.-K. R. Choo, "Toward detection and attribution of cyber-attacks in IoT-enabled cyber–physical systems," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13712–13722, Sep. 2021.

[5] B. B. Zarpelão et al., "A survey of intrusion detection in Internet of Things," *J. Netw. Comput. Appl.*, 2017.

[6] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Netw.*, vol. 8, no. 3, pp. 26–41, May 1994.

[7] R. Yasaei, F. Hernandez, and M. A. Al Faruque, "IoT-CAD: Context-aware adaptive anomaly detection in IoT systems through sensor association," in *Proc. 39th Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[8] M. A. Faruque, F. Regazzoni, and M. Pajic, "Design methodologies for securing cyber-physical systems," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis (ISSS)*, 2015, pp. 30–36.

[9] M. Zaman and C.-H. Lung, "Evaluation of machine learning techniques for network intrusion detection," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp. (NOMS)*, 2018, pp. 1–5.

[10] S. Kumar, "Survey of current network intrusion detection techniques," *Washington Univ. St. Louis*, 2007, pp. 1–18.

[11] A. Barua, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque, "Hierarchical temporal memory-based one-pass learning for real-time anomaly detection and simultaneous data prediction in smart grids," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 3, pp. 1770–1782, May 2020.

[12] K. A. Da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. De Albuquerque, "Internet of Things: A survey on machine learning-based intrusion detection approaches," *Comput. Netw.*, vol. 151, Mar. 2019, pp. 147–157.
[13] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of SVM and ANN for intrusion detection," *Comput. Oper. Res.*, vol. 32, no. 10, pp. 2617–2634, 2005.
[14] R.-C. Chen, K.-F. Cheng, and C.-F. Hsieh, "Using rough set and support vector machine for network intrusion detection," 2010, *arXiv:1004.0567*.
[15] D. Jing and H.-B. Chen, "SVM based network intrusion detection for the UNSW-NB15 dataset," in *Proc. 13th Int. Conf. ASIC ASICON*, 2019, pp. 1–4.
[16] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.
[17] Y. R. Siwakoti et al., "Advances in IoT security: Vulnerabilities, enabled criminal services, attacks and countermeasures," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11224–11239, Jul. 2023.
[18] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[19] J. Wang, L. Chen, and M. A. Al Faruque, "DOMINO: Domain-invariant Hyperdimensional classification for multi-sensor time series data," in *Proc. 42nd ACM/IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.
[20] J. Wang, H. Chen, M. Issa, S. Huang, and M. Imani, "Late breaking results: Scalable and efficient Hyperdimensional computing for network intrusion detection," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–2.
[21] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proc. Int. Symp. Low Power Electron. Design, (ISLPED)*, 2016, pp. 64–69.
[22] J. Wang, S. Huang, and M. Imani, "DistHD: A learner-aware dynamic encoding method for Hyperdimensional classification," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[23] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proc. Int. Conf. High Perform. Comput., Netw., Stor. Anal. (SC)*, 2021, pp. 1–15.
[24] B. B. Andersen, H. J. G. Gundersen, and B. Pakkenberg, "Aging of the human cerebellum: A stereological study," *J. Comp. Neurol.*, vol. 466, no. 3, pp. 356–365, 2003.
[25] B. Pakkenberg et al., "Aging and the human neocortex," *Exp. Gerontol.*, vol. 38, nos. 1–2, pp. 95–99, 2003.
[26] J. Wang and M. A. A. Faruque, "Robust and scalable Hyperdimensional computing with brain-like neural adaptations," 2023, *arXiv:2311.07705*.
[27] J. Asharf, N. Moustafa, H. Khurshid, E. Debie, W. Haider, and A. Wahab, "A review of intrusion detection systems using machine and deep learning in Internet of Things: Challenges, solutions and future directions," *Electronics*, vol. 9, no. 7, p. 1177, 2020.
[28] A. Bivens, C. Palagiri, R. Smith, and B. Szymanski, "Network-based intrusion detection using neural networks," in *Proc. Intell. Eng. Syst. Artif. Neural Netw.*, 2002, pp. 579–584.
[29] D. Chou and M. Jiang, "A survey on data-driven network intrusion detection," in *Proc. ACM Comput. Surveys*, 2021, pp. 1–36.
[30] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. Symp. Comput. Intell. Security Defense Appl. (CISDA)*, 2009, pp. 1–6.
[31] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (CIS)*, 2015, pp. 1–6.
[32] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Security Privacy (ICISSP)*, 2018, pp. 1–9.
[33] J. L. Leevy and T. M. Khoshgoftaar, "A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 big data," *J. Big Data*, vol. 7, p. 104, Nov. 2020.
[34] A. Rosay et al., "Multi-layer perceptron for network intrusion detection," *Ann. Telecommun.*, vol. 77, pp. 371–394, Jun. 2022.
[35] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.
[36] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.
[37] P. Poduval et al., "GrapHD: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Front. Neurosci.*, vol. 16, Feb. 2022, Art. no. 757125.
[38] M. Imani et al., "A framework for collaborative learning in secure high-dimensional space," in *Proc. IEEE 12th Int. Conf. Cloud Comput.*, 2019, pp. 435–446.
[39] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2021, pp. 56–61.
[40] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artif. Intell. Rev.*, vol. 55, pp. 4523–4555, Aug. 2022.
[41] B. Komer, T. C. Stewart, A. R. Voelker, and C. Eliasmith, "A neural representation of continuous space using fractional binding," in *Proc. 41st Annu. Meet. Cogn. Sci. Society*, 2019, pp. 2038–2043.
[42] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. 20th Int. Conf. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1177–1184.
[43] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*. Berlin, Germany: Springer, 2006, pp. 75–102.
[44] A. Rosay, F. Carlier, and P. Leroux, "MLP4NIDS: An efficient MLP-based network intrusion detection for CICIDS2017 Dataset," in *Proc. 2nd Int. Conf. Mach. Learn. Netw.*, 2019, pp. 240–254.
[45] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011.
[46] J. Rapin and O. Teytaud. "Nevergrad-a gradient-free optimization platform." 2018. [Online]. Available: https://GitHub.com/FacebookResearch/Nevergrad
[47] V. Kathail, "Xilinx vitis unified software platform," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2020, pp. 173–174.

Junyao Wang (Graduate Student Member, IEEE) received the B.S. degree in mathematics and statistics and the M.S. degree in operations research from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 2019 and 2020, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California at Irvine, Irvine, CA, USA.

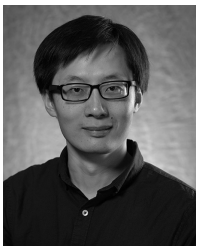
Her research primarily includes resource-efficient machine learning algorithms, applications of graph neural networks in autonomous systems, and the intersection of machine learning and sensor fusion.

Haocheng Xu (Graduate Student Member, IEEE) received the B.S. degree from the College of Information and Electrical Engineering, China Agricultural University, Beijing, China, in 2017, and the M.S. degree from the Viterbi School of Engineering, University of Southern California, Los Angeles, CA, USA, in 2019. He is currently pursuing the Ph.D. degree in computer engineering with the University of California at Irvine, Irvine, CA, USA.

His research interests include efficient machine learning, deep learning accelerators, and AI/ML systems.

Yonatan Gizachew Achamyeleh received the B.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2021. He is currently pursuing the Ph.D. degree in computer engineering with the University of California at Irvine, Irvine, CA, USA.

He has previously interned with Siemens Technology, Princeton, NJ, USA, and Intel Labs, Hillsboro, OR, USA. He is a member of the Autonomous and Intelligent Cyber-Physical Systems Lab. His research focuses on the security aspects of embedded and cyber–physical systems, especially in sectors, such as manufacturing, IoT, and healthcare.



Sitao Huang (Member, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, in 2014, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 2017 and 2021, respectively.

He is an Assistant Professor with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA, USA. His research interests include hardware accelerators, compilers for accelerators, and heterogeneous systems.

Dr. Huang is a 2022 DARPA Forward Riser. His research won the Best Paper Award at IDEAL 2021, the Best Paper Nomination at ASP-DAC 2021, and the Student Innovation Award at the 2018 IEEE HPEC Graph Challenge.

Mohammad Abdullah Al Faruque (Senior Member, IEEE) received the Ph.D. degree in computer science from Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2009.

He is currently with the University of California at Irvine, Irvine, CA, USA, as a Full Professor and Directing the Embedded and Cyber-Physical Systems Lab, where he also directs the Samueli School of Engineering Autonomous Systems Initiatives. His research focuses on the system-level design of embedded and cyber–physical systems (CPS) with a special interest in low-power design, CPS security, and data-driven CPS design.

Prof. Al Faruque has received four Best Paper Awards (ACSAC 2022, DATE 2016, DAC 2015, and ICCAD 2009). He also received the IEEE Technical Committee on Cyber-Physical Systems Early-Career Award and the IEEE CEDA Ernest S. Kuh Early Career Award. He has been awarded the Thomas Alva Edison Patent Award for one of his inventions. He is an ACM Senior Member. He is also the IEEE CEDA Distinguished Lecturer.