TrafficHD: Efficient Hyperdimensional Computing for Real-Time Network Traffic Analytics

Haodong Lu, Zhiyuan Ma, Xinran Li, Shiyan Bi, Xiaoming He, and Kun Wang†
School of Microelectronics, Fudan University, Shanghai, China
ihaodonglu@gmail.com, kun.wang@ieee.org

ABSTRACT

With the evolution of network infrastructure, the pattern of network traffic becomes unprecedentedly complex. Conventional machine learning algorithms struggle to cope with the high-dimensional data and real-time processing speeds required in such complex networks. Fortunately, Hyperdimensional Computing (HDC), which is power-efficient and supports parallel processing, provides a potential solution to this challenge. In this paper, we present *TrafficHD*, a novel classification framework that leverages HDC to analyze network traffic in real-time. By transforming network traffic features into high-dimensional binary vectors, *TrafficHD* enables the rapid execution of recognition tasks within the constraints of real-time systems. Extensive evaluations on a wide range of network tasks show that *TrafficHD* is 30.57× and 98.32× faster than state-of-the-art (SOTA) machine learning and HDC algorithms while providing 3× higher robustness to network noise.

ACM Reference Format:
Haodong Lu, Zhiyuan Ma, Xinran Li, Shiyan Bi, Xiaoming He, and Kun Wang†. 2024. TrafficHD: Efficient Hyperdimensional Computing for Real-Time Network Traffic Analytics. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657330

†Corresponding author
This work was financially supported in part by National Key Research and Development Program of China under Grant 2021YFA1003602, and in part by Shanghai Pujiang Program under Grant 22PJD003.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657330

1 INTRODUCTION

With the increase in network traffic, the transfer of data across digital infrastructure has become more prominent. Consequently, network traffic analysis enables the robust characterization of traffic profiles and detection of security threats [1], facilitating intelligent network management, Quality of Service (QoS) optimization, and security measures. However, as the volume and velocity of network traffic increase, traditional classification mechanisms seek advanced solutions to preserve efficiency and security.

Previous work has extensively explored machine learning-based methods for analyzing network traffic [13]. Unfortunately, these existing methods suffer from limitations including high computational costs and the inability to process and classify traffic in real-time [8]. In addition, the requirement for extensive training and the difficulty of adapting to novel traffic patterns hinder the practicality of machine learning approaches in dynamic network environments.

Hyperdimensional Computing (HDC) is an emerging computational paradigm that draws inspiration from the human brain to process information in high-dimensional spaces [5, 7]. HDC encodes low-dimensional input to hypervector, typically on the order of 10^4 dimensions, to perform various learning tasks. Compared to traditional machine learning algorithms, HDC offers three distinct advantages for network traffic analysis: (1) **Robustness to Noise.** The inherent error tolerance of hyperdimensional space allows HDC to handle noisy and unstable network traffic data. (2) **Scalability.** HDC is utilized to learn from minimal data samples in communication networks. (3) **High-level Parallelism.** The operations in HDC are inherently parallelized and can be executed rapidly, which is suitable for real-time data processing.

Despite the success of HDC, analyzing network traffic presents unique challenges that need to be addressed. The first challenge involves efficiently processing the vast and diverse data streams that characterize network traffic without causing model saturation. The second challenge is rapidly processing network traffic, which is difficult when using HDC on traditional hardware platforms such as CPUs and GPUs [6]. These hardware architectures lack full optimization for parallel bitwise operations in HDC, resulting in a gap in harnessing theoretical efficiency gains.

In this paper, we propose *TrafficHD*, a novel HDC learning framework for highly efficient and robust network traffic analysis. To achieve high recognition accuracy in low dimensions, we redesign the learning framework using adaptive updating and dynamic encoding. Furthermore, to achieve real-time network traffic analysis, we accelerate model inference through software-hardware co-optimization and create customized circuits based on FPGA. The main contributions of this paper can be summarized as follows:

(1) To the best of our knowledge, *TrafficHD* is the first work to apply HDC for real-time network traffic analysis. Compared to conventional HDC algorithms using static encoders, *TrafficHD* adopts a dynamic HDC encoding methodology that identifies and regenerates insignificant dimensions.

(2) We also conduct algorithm-hardware co-design at the inference stage to achieve network traffic analysis in real-time. This involves the creation of a hardware-friendly encoding module to reduce computation costs. Furthermore, by decomposing cosine similarity and streamlining redundant processes, we improve the speed of association searches while minimizing on-chip resource consumption.

(3) We evaluate the performance of our method across different tasks, *i.e.*, traffic classification and network attack detection. Experimental results show that *TrafficHD* achieves an average accuracy improvement of 10.3% over state-of-the-art (SOTA) machine learning algorithms and 15.2% over SOTA HDC methods. For real-time network traffic detection, *TrafficHD* achieves speedups of 30.57× and 98.32× compared to conventional approaches.

2 HDC PRELIMINARIES

HDC draws inspiration from the human brain that processes information in a high-dimensional space, with massive neurons and synapses contributing to its computational power. In HDC, inputs are mapped into high-dimensional spaces, termed *hypervectors*, with each dimension generated randomly. A distinctive attribute of this hyperdimensional space is the existence of quasi-orthogonal hypervectors, which facilitate highly parallel operations including similarity calculations, bundling, and binding.

From a mathematical perspective, for random bipolar hypervectors $\mathcal{H}_1$ and $\mathcal{H}_2$ with dimension D, *i.e.*, $\mathcal{H}_1, \mathcal{H}_2 \in \{-1, 1\}^{\mathcal{D}}$, the dot product $\mathcal{H}_1 \cdot \mathcal{H}_2 \approx 0$ as D becomes sufficiently large. (1) **Similarity** involves computing the distance between a query hypervector and a class hypervector, denoted as $\delta(\cdot, \cdot)$. The cosine similarity, a prevalent metric for real-valued hypervectors, is defined as,

$$\delta\left(\mathcal{H}_1, \mathcal{H}_2\right) = \frac{\mathcal{H}_1 \cdot \mathcal{H}_2}{\|\mathcal{H}_1\| \cdot \|\mathcal{H}_2\|}, \tag{1}$$

where $\mathcal{H}_1 \cdot \mathcal{H}_2$ is the dot product. $\|\mathcal{H}_1\|$ and $\|\mathcal{H}_2\|$ represent the Euclidean norms of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. (2) **Binding** merges two hypervectors via element-wise multiplication, generating a new hypervector, *i.e.*, $\mathcal{H}_{\text{bind}} = \mathcal{H}_1 * \mathcal{H}_2$. The resulting hypervector, $\mathcal{H}_{\text{bind}}$, is orthogonal or nearly orthogonal to the original hypervectors, *i.e.*, $\delta\left(\mathcal{H}_{\text{bind}}, \mathcal{H}_1\right) \simeq 0$ and $\delta\left(\mathcal{H}_{\text{bind}}, \mathcal{H}_2\right) \simeq 0$. (3) **Bundling** combines multiple hypervectors via element-wise addition, forming a single hypervector that maintains a degree of similarity to each original hypervector, namely, $\mathcal{H}_{\text{bundle}} = \mathcal{H}_1 + \mathcal{H}_2$. This operation generates composite representations and provides an easy way to check the existence of a query hypervector in a bundled set. Specifically, $\delta\left(\mathcal{H}_{\text{bundle}}, \mathcal{H}_1\right) \gg 0$, while $\delta\left(\mathcal{H}_{\text{bundle}}, \mathcal{H}_3\right) \simeq 0$ $(\mathcal{H}_3 \neq \mathcal{H}_1, \mathcal{H}_2)$.

3 *TRAFFICHD* ADAPTIVE LEARNING

3.1 Overview

Fig. 1 illustrates the training procedure of *TrafficHD*. Initially, *TrafficHD* encodes traffic features into hypervectors utilizing the encoding module (❶). The similarity of the query hypervector over each class is calculated and the class with the highest similarity is selected (❷). *TrafficHD* then conducts adaptive updating of the class hypervectors (❸). The details of adaptive updating are elaborated in Sec. 3.3. Following a single-pass through all training datasets, the class hypervector adapts to the patterns of its corresponding category. To further improve the classification accuracy, *TrafficHD* identifies dimensions with the lowest impact on accuracy (❹). Subsequently, *TrafficHD* drops insignificant dimensions from the base hypervector within the encoding module and reconstructs hypervectors in these dimensions for subsequent training iterations (❺).

3.2 Non-linear Encoding

Several encoding methodologies proposed in the literature [4, 11] linearly map individual features into a hyperspace. However, to capture the non-linear relationships inherent in network traffic data, we exploit the Radial Basis Function (RBF) kernel [12] to design a non-linear encoding module. Considering a network traffic vector with n features, denoted by $\vec{F} = \{f_1, f_2, \ldots, f_n\}$, the encoding module transforms this feature vector into a hypervector $\vec{H} = \{h_1, h_2, \ldots, h_D\}$ with D dimensions. This transformation is achieved by determining each encoded dimension by multiplying the cosine and sine of the dot product between the input feature vector and a randomly generated vector. Specifically, this procedure is expressed as $h_i = \cos\left(\vec{B}_i \cdot \vec{F} + b\right) \times \sin\left(\vec{B}_i \cdot \vec{F}\right)$, where $\vec{B}_i$ follows Gaussian distribution (μ=0 and σ=1) and b represents a random bias uniformly sampled from $[0, 2\pi]$. After this step, each element h_i in the hypervector $\vec{H}$ contains a non-binary value. In contrast to floating-point representations [15], HDC typically utilizes binary (bipolar) hypervectors to

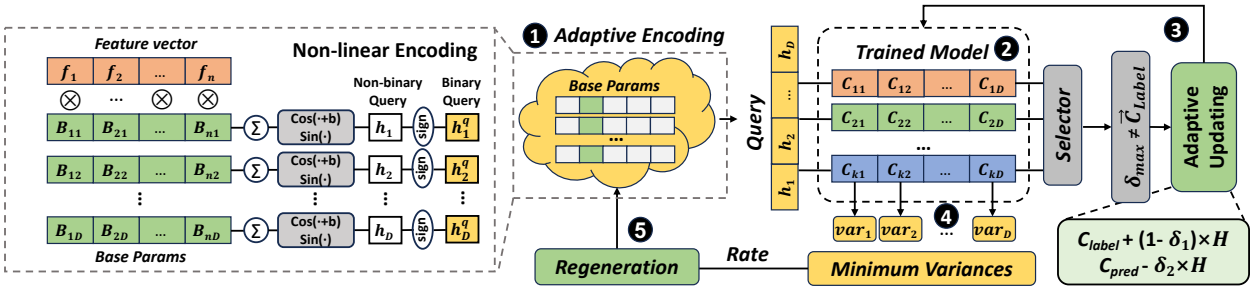


Figure 1: Workflow of *TrafficHD* adaptive learning.

improve computational efficiency. Thus, we quantize the final encoded hypervector by binarizing it with a sign function. The overall encoding process is shown in Fig. 1 (❶).

3.3 Adaptive Updating

In HDC-based methodologies, simplistic hypervector addition may result in the saturation of class hypervectors due to the predominance of data samples exhibiting common patterns. This saturation obscures information with less common features stored in class hypervectors. As a result, data samples with non-common features are likely to be misclassified by the model. To address this issue, *TrafficHD* exploits adaptive updating for efficient and accurate HDC learning.

In each training iteration, *TrafficHD* refines the process by recognizing and eliminating common patterns to prevent saturation in each class hypervector. To be specific, *TrafficHD* progressively integrates each encoded sample into the corresponding category hypervector, based on the new information contributed by the sample. If an encoded sample is correctly classified, the update is skipped. Otherwise, *TrafficHD* adds a portion of the query hypervector to the corresponding class hypervector and subtracts some patterns from the misclassified class, thus effectively reducing the risk of model saturation. Assuming that the new misclassified training sample is encoded as $\overrightarrow{\mathcal{H}}$, *TrafficHD* computes the cosine similarity of $\overrightarrow{\mathcal{H}}$ with each class hypervector $\overrightarrow{C}$ using Eq. 1. If the input data belongs to label l, and the most similar class determined is label l', then the model updates as follows,

$$\begin{aligned} \overrightarrow{C_l} &\leftarrow \overrightarrow{C_l} + \eta \times (1 - \delta_l) \times \overrightarrow{\mathcal{H}}, \\ \overrightarrow{C_{l'}} &\leftarrow \overrightarrow{C_{l'}} - \eta \times \delta_{l'} \times \overrightarrow{\mathcal{H}}, \end{aligned} \tag{2}$$

where η denotes the learning rate. A large value of δ_l suggests that the model already contains the incoming data point. To mitigate model saturation, merely a minimal part of the encoded query is integrated, leading to $(1 - \delta_l)$ nearly zero. Conversely, a low value of δ_l signifies the emergence of a novel pattern not previously captured by the model, necessitating a more substantial update, *i.e.*, $(1 - \delta_l \approx 1)$.

3.4 Dimension Regression

In the training phase, each class is represented by a single hypervector. The goal of HDC training is to store information that represents the common patterns of each class. However, network traffic exhibits similar patterns, which leads to weak classifiers that cannot distinguish between the patterns of different classes. This phenomenon results in the query having close similarity values with several classes. To address this issue, we can identify dimensions that have minimal impact on the overall classification accuracy. Specifically, we compute the variance of each dimension across all class hypervectors after each training iteration, as shown in Fig. 1 (❹). A low variance in specialized dimensions indicates the storage of common information, thereby failing to aid in distinguishing between class patterns.

Instead of removing dimensions with low variance, *TrafficHD* regenerates the base parameters of the dimensions corresponding to the encoding module, as illustrated in Fig. 1 (❺). Utilizing the calculated variances, *TrafficHD* selects $r\%$ of the dimensions with the lowest variance as candidate regeneration dimensions, where r represents the predefined regeneration rate before training. After that, *TrafficHD* regenerates those dimensions that can improve variance, thereby increasing the final classification accuracy.

4 *TRAFFICHD* INFERENCE

Once the model is converged, *TrafficHD* stores hypervectors that capture the unique patterns of each category. These hypervectors are then used for subsequent inference tasks. The inference process involves two main computations: feature encoding and associative search. During inference, *TrafficHD* utilizes the same encoding module from the training phase to map the feature vector to the query hypervector. Then, it measures the similarity between the class hypervectors and the query hypervector. The category exhibiting the highest similarity with the query hypervector is thereby identified as the matching category.

To achieve real-time analysis of network traffic, the execution process of the classifier needs to be simplified so that it can be processed in real-time on resource-constrained devices. However, the similarity computation consists of a

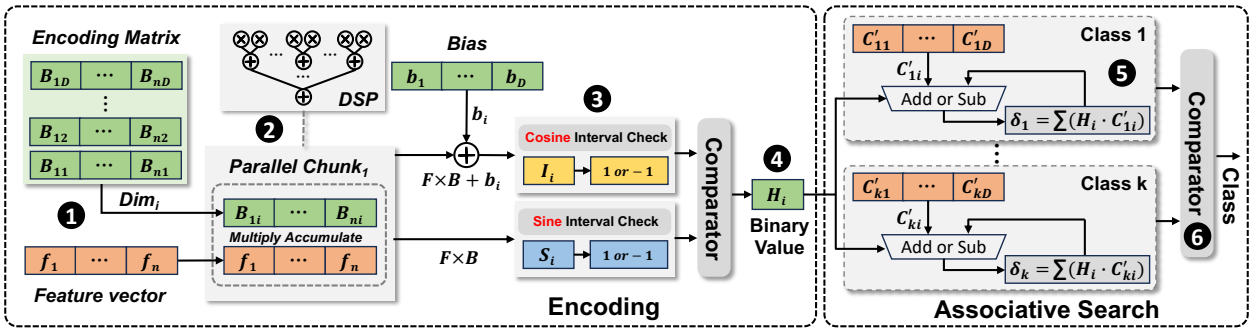


Figure 2: Hardware acceleration of the encoding and associative search.

high-dimensional dot product between query hypervector $\mathcal{H}$ and class hypervectors C_i. According to Eq. 1, this process can be segmented into three components: $\mathcal{H} \cdot C_i$, $\mathcal{H} \cdot \mathcal{H}$, and $C_i \cdot C_i$. When comparing the query hypervector to all classes, the term $\mathcal{H} \cdot \mathcal{H}$ remains constant and therefore can be omitted. As for $C_i \cdot C_i$, since C_i remains constant during the inference stage, it can be pre-normalized and used in the following inference. Consequently, the computation of cosine similarity can be simplified to $\delta(\mathcal{H}, C_i) = \mathcal{H} \cdot C_i$, effectively reducing computational cost by 1/3.

5 HARDWARE ACCELERATION

HDC can be implemented on traditional hardware platforms, such as CPU, GPU, and FPGA. Considering HDC involves high-parallel bit-level operations, FPGA is a suitable candidate for efficient inference acceleration. Fig. 2 illustrates the FPGA-based implementation of *TrafficHD* inference, which consists of encoding and associative search accelerations.

5.1 Encoding Acceleration

The first step in inference is to encode the input features to a binary hypervector (Fig. 2 ❶-❹). However, this requires vector-matrix multiplication and computation of trigonometric functions, which predominates the inference time. To speed up this process, *TrafficHD* offers two hardware-level optimizations. The first optimization involves segmenting the encoding parameters into multiple chunks, which are then computed with the input features (❷). Each chunk utilizes Digital Signal Processing (DSP) to perform parallel multiply-accumulate operations on n input features. Moreover, the computation result within each chunk is seamlessly transmitted to the subsequent modules. This approach helps to harness the inherent parallel computing attributes of HDC.

After obtaining the encoded value, the next step involves the computation of the trigonometric functions, which can be implemented by CORDIC IP. However, directly computing the trigonometric function with D-dimensional points is computationally intensive. Given the binary encoding of the hypervector, we simplify the calculation through an interval check for trigonometric functions (❸). Specifically, leveraging the periodicity of trigonometric functions, the output sign can be inferred directly from the interval location of the input value. Taking cosine as an example, if the input data falls in the range $[0, \pi/2]$, the cosine output is positive. Conversely, if it lies within the range $[\pi/2, 3\pi/2]$, the output is negative. This procedure reduces the real number calculation to sign bit determination. Finally, the encoded hypervector is obtained by performing a bitwise comparison of the cosine and sine outputs (❹). The query hypervector can be stored in FPGA using a single bit for each dimension represented in binary $\{0, 1\}$.

Table 1: Datasets (n: features size, k: number of classes).

	n	k	Data Size	Description
Moore	248	12	360,332	Network traffic classification [9]
ISCXTor2016	28	8	8,044	Tor traffic detection [2]
CIC-IoT	46	8	238,687	IoT attack detection [10]
NSL-KDD	122	2	125,972	Network intrusion detection [14]

5.2 Associative Search Acceleration

The associative search computes the similarity between the query hypervector and each class. To reduce the cost of cosine similarity in inference, we adopt a simplification strategy as described in Sec. 4. This involves normalizing the class hypervectors beforehand and omitting the normalization of the query hypervector as it is a constant term for all classes. We also employ binary encoding to eliminate the multiplication operations required in associative search. For each encoded query, we prefetch each class hypervector and flip the sign bit of each element according to the query bits (❺). Then, a sequential accumulation of all elements yields the results for each class. Finally, a comparator at the end of the associative search determines the class with the highest similarity (❻).

6 EVALUATION

6.1 Experimental Setup

We evaluate the performance of *TrafficHD* using an Intel Xeon Silver 4210R CPU and Xilinx Kintex-7 XC7K325T FPGA on four real-world traffic datasets. For inference, the functionality of *TrafficHD* is designed and synthesized using Vivado 2020.1. The validation dataset is summarized in Tab. 1, and

the datasets are divided into training and test sets at 80% and 20%, respectively. We measure system performance in terms of algorithm accuracy, execution time, and energy efficiency, comparing the CPU-FPGA platform results with CPU and GPU platforms.

State-of-the-art Algorithms. We compare *TrafficHD* with SOTA machine learning algorithms used in network traffic, including Support Vector Machine (SVM) and Deep Neural Network (DNN). The SVM is trained with the scikit-learn, and the DNN is trained with the PyTorch. We also compare *TrafficHD* with SOTA HDC approach [3] that employs static encoding, and report the results for both single-pass and iterative training. In single-pass learning, the model is trained only once on the dataset. The hypervector dimension for *TrafficHD* and baseline HD is adopted as D=1,000 and D=4,000, respectively. For iterative training, we perform 20 iterations and regenerate the encoding parameters with the lowest 10% variance in each iteration.

6.2 Software Evaluation

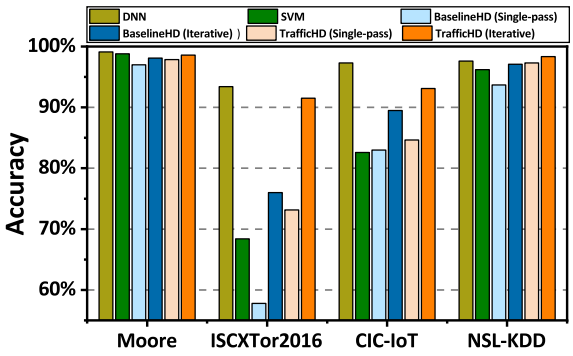
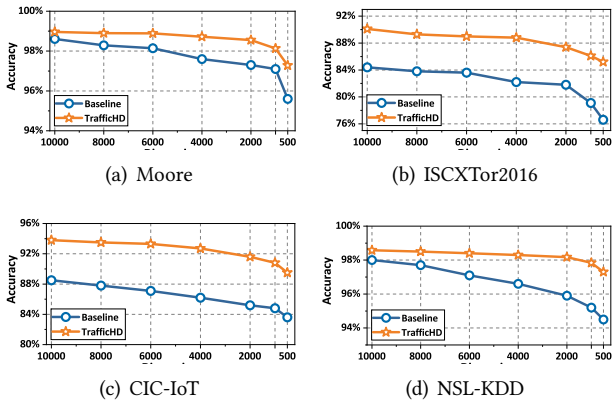


Figure 3: Classification accuracy of *TrafficHD* and state-of-the-art algorithms.

6.2.1 ***TrafficHD Accuracy.*** Fig. 3 compares *TrafficHD* classification accuracy with SOTA algorithms. For HDC-based methods, we examine two training strategies, *single-pass training* and *iterative training*. In terms of *TrafficHD*, the single-pass training method involves incorporating dynamic updates into the dataset without allowing dimension regeneration. As shown in Fig. 3, *TrafficHD* achieves similar accuracy with SOTA DNNs and 10.3% higher accuracy than SVM on average. Additionally, *TrafficHD* delivers on average 15.2% and 5.8% higher accuracy than the baseline HD (Single-pass) and baseline HD (Iterative), respectively. These results indicate that *TrafficHD* achieves higher accuracy than SOTA HD while reducing dimensions by 4× on average. *TrafficHD* achieves this performance by enabling adaptive encoding and adaptive training, thereby mitigating model saturation.

6.2.2 ***Dimension Impact.*** HDC operates by manipulating vector patterns in high-dimensional spaces, where the number of dimensions required depends on the specific dataset.



(a) Moore (b) ISCXTor2016

(c) CIC-IoT (d) NSL-KDD

Figure 4: Impact of dimension reduction.

However, reducing the dimensions can speed up computation during both the training and inference stages. Fig. 4 demonstrates the impact of dimension reduction on classification accuracy in various network scenarios. The results indicate that while reducing dimensions generally leads to accuracy loss for both approaches, *TrafficHD* consistently outperforms baseline HD, maintaining higher accuracy even at reduced dimensions. For ISCXTor2016 and CIC-IoT datasets, *TrafficHD* using D=500 can achieve similar accuracy as full dimensionality in baseline HD. *TrafficHD* achieves such high accuracy at low dimensions due to its ability to regenerate insignificant dimensions.

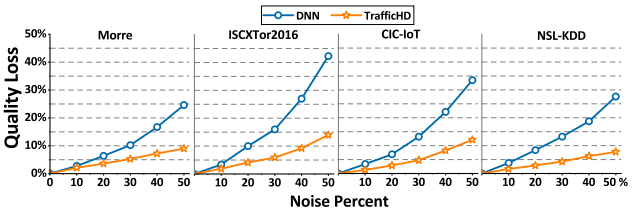


Figure 5: Robustness of DNN and *TrafficHD*.

6.2.3 ***Robustness to Noise.*** Fig. 5 shows the classification accuracy of DNN and *TrafficHD* when random noise is introduced into traffic packets. The results demonstrate that *TrafficHD* possesses significant robustness to noise compared to DNN. This robustness is attributed to the holographic mapping technique, which distributes feature information across all dimensions of the encoded data. As a result, when noise is introduced into traffic data, it only fails a portion of each hypervector, without losing the entire information. In DNNs, noise in input features can significantly increase model error during forward propagation, severely affecting the overall accuracy. For example, after adding 50% noise to the ISCXTor2016 dataset, the accuracy of the DNN model drops by up to 42.3%, while the maximum quality loss using *TrafficHD* is only 14.1%.

6.3 Hardware Evaluation

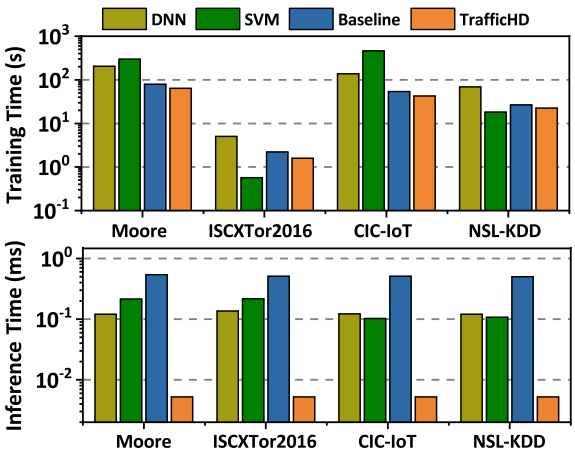


Figure 6: Training and inference efficiency of *TrafficHD* and SOTA algorithms (log scale).

6.3.1 ***TrafficHD Execution Time.*** Fig. 6 evaluates the efficiency of *TrafficHD* with SOTA machine learning algorithms and HDC-based methods. Both DNN and HDC-based methods are trained on GPU, except for SVM which is executed on CPU. The results show that the HDC-based method can quickly accomplish model training by fusing hyperdimensional information compared to DNN. Additionally, the HDC-based method can be further accelerated by parallel vector-matrix operations during the training procedure. With fewer iterations and data dimensions, *TrafficHD* can achieve faster model training than baseline HD.

For model inference, both *TrafficHD* and baseline HD represent class hypervectors in a 16-bit fixed-point, while DNN parameters are represented using an 8-bit fixed-point. Since *TrafficHD* processes input features in parallel, its inference cycle is proportional to the dimension D. As shown in Fig. 6, *TrafficHD* provides an average 23.84×, 30.57×, and 98.32× speedup compared to SOTA DNN, SVM, and baseline HD, respectively. *TrafficHD* can achieve low inference latency as it requires significantly lower dimensions. Moreover, FPGAs allow for parallel encoding and associative search, effectively utilizing the inherent parallelism in HDC.

6.3.2 ***Resource Utilization.*** Tab. 2 shows the resource utilization of the FPGA implementation with hypervector size D=1,000. The encoding and associative modules are designed to use different hardware resources depending on the feature size n and the number of classes k. In detail, the encoding module uses DSPs that match the feature size n to enable parallel processing of input features. We also obtain that the core computing module of the Xilinx Kintex-7 XC7K325T accelerator card consumes less than 1.5 W. This makes it an ideal choice for network traffic analysis on resource-constrained edge devices.

Table 2: Resources utilization of *TrafficHD*.

Parameters (n, k)	(248, 12)	(28, 8)	(46, 8)	(122, 2)
LUTs	10694	1651	2388	5271
FF	8636	1405	2125	4326
BRAM	173	31	57	141
DSP	248	28	46	122

7 CONCLUSION

In this paper, we propose a novel framework named *TrafficHD* to address two prevalent challenges in HDC-based traffic analysis, *i.e.*, model saturation and execution delay. *TrafficHD* exploits adaptive learning combined with dimension regeneration to eliminate common traffic patterns and improve classification accuracy. Moreover, *TrafficHD* optimizes the computational process through mathematical simplifications. To realize real-time network traffic analysis, *TrafficHD* is deployed on an FPGA platform with custom optimization. Our evaluation shows that *TrafficHD* achieves an average 15.2% improvement in accuracy and delivers a 98.32× speedup compared to SOTA algorithms.

REFERENCES

[1] Oluwamayowa Ade Adeleke et al. 2022. Network traffic generation: A survey and methodology. *ACM Comput. Surv.* (Jan. 2022).
[2] Arash Habibi Lashkari et al. 2017. Characterization of tor traffic using time based features. In *ICISSP*.
[3] Hernández-Cano et al. 2021. OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system. In *DATE*.
[4] Mohsen Imani et al. 2018. Hierarchical hyperdimensional computing for energy efficient classification. In *DAC*.
[5] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* (2009).
[6] Behnam Khaleghi et al. 2022. GENERIC: Highly efficient learning engine on edge using hyperdimensional computing. In *DAC*.
[7] Denis Kleyko et al. 2022. A survey on hyperdimensional computing aka vector symbolic architectures, part I: Models and data transformations. *ACM Comput. Surv.* (Dec. 2022).
[8] Manish Marwah and Martin Arlitt. 2022. Deep learning for network traffic data. In *KDD*.
[9] Andrew W. Moore and Denis Zuev. 2005. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS*.
[10] Euclides Carlos Pinto Neto et al. 2023. CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment. *Sensors* (June 2023).
[11] Abbas Rahimi et al. 2016. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *ISLPED*.
[12] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *NIPS*.
[13] Meng Shen et al. 2023. Machine learning-powered encrypted network traffic analysis: A comprehensive survey. *IEEE Commun. Surveys Tuts* (Jan. 2023).
[14] Mahbod Tavallaee et al. 2009. A detailed analysis of the KDD CUP 99 data set. In *CISDA*.
[15] Chen Wu et al. 2021. Low-precision floating-point arithmetic for high-performance FPGA-based cnn acceleration. *ACM Trans. Reconfigurable Technol. Syst.* (Nov. 2021).