

# An Efficient and Scalable Hyperdimensional Computing Framework for Anomaly Classification in Industrial Systems

Víctor Ortega<sup>a</sup>, Soledad Escolar<sup>b</sup>, Fernando Rincón<sup>c</sup>, Jesús Barba<sup>d</sup>, Julián Caba<sup>e</sup>  
and Juan Carlos López<sup>f</sup>

*School of Computer Science, University of Castilla-La Mancha, Ciudad Real, Spain*

**Keywords:** Hyperdimensional Computing, Industrial Systems, Anomaly Detection, Predictive Maintenance, Machine Learning.

**Abstract:** This paper presents a hyperdimensional computing (HDC)-based framework for anomaly classification, designed to meet the specific demands of industrial systems. Inspired by cognitive processes, HDC employs high-dimensional representations to enable robust, low-complexity, and hardware-efficient computation. The proposed framework encompasses the entire processing pipeline, from data encoding to anomaly classification, and is optimized for efficient execution on both conventional computing platforms and resource-constrained devices. To assess its effectiveness, we conduct a case study based on a real-world scenario involving 118 emergency lighting devices that collect and transmit operational data to a central sink capable of detecting anomalous behavior. Experimental results demonstrate that the proposed approach achieves high classification accuracy and confirm its suitability for deployment in integrated industrial systems with limited computational resources.

## 1 INTRODUCTION

Fault detection and mitigation is an ongoing challenge that has become critically important in modern industrial environments, where intelligent systems and interconnected devices are increasingly integrated into complex production and monitoring infrastructures across a wide range of applications. A random, non-compliant, unexpected behavior in a system, which may even be malicious is called an anomaly (Baddar et al., 2014). The root cause of such deviating behavior can vary widely, e.g. component failures, software errors and environmental factors. Regardless the root cause, the consequences of not having a proper anomaly detection strategy can be serious, making timely and effective mitigation essential.

Anomaly detection aims to identify patterns in events that deviate significantly from expected or normal behavior. Typically, each event is represented by

a data instance composed of a set of descriptive features, which are not necessarily all equally informative (Bhuyan et al., 2014). The problem may be formulated as a classification problem in which anomalous behaviors need to be distinguished from normal behaviors (Baddar et al., 2014). The literature on anomaly detection in industrial systems encompasses a broad spectrum of approaches across various research areas (Chandola et al., 2009; Baddar et al., 2014), such as: statistics, Machine Learning (ML), data mining, information theory, and spectral theory. Among these approaches, those based on ML have received increased attention for their ability to learn from data and effectively classify or group abnormal and previously unseen behaviors (Nassif et al., 2021). However, as noted in (Hernández-Cano et al., 2021), several limitations remain, primarily concerning the efficiency, power consumption, and robustness of the algorithms. Specifically, ML-based solutions are highly computationally intensive, making them impractical for edge devices, which often operate online and under strict constraints in memory, processing power, and energy consumption. Furthermore, ML algorithms often exhibit high sensitivity to noise and hardware failures and struggle to establish causality, meaning that they cannot directly

<sup>a</sup>  <https://orcid.org/0009-0003-1548-3260>

<sup>b</sup>  <https://orcid.org/0000-0002-8019-9640>

<sup>c</sup>  <https://orcid.org/0000-0003-4688-8650>

<sup>d</sup>  <https://orcid.org/0000-0003-1931-3245>

<sup>e</sup>  <https://orcid.org/0000-0002-7641-4643>

<sup>f</sup>  <https://orcid.org/0000-0002-7372-1568>

determine whether one variable influences changes in another. This limitation is particularly critical in anomaly detection, where the goal is not only to identify anomalous behaviors but also to understand their root causes and provide meaningful justifications for effective remediation. The work presented in (Nassif et al., 2021) offers a systematic literature review of ML-based techniques to anomaly detection in Internet of Things (IoT) environments, categorizing them into supervised, semi-supervised, and unsupervised methods.

Hyperdimensional Computing (HDC) is a brain-inspired computational paradigm designed to address classification problems by mimicking aspects of cognitive processing. HDC operates on high-dimensional vectors, also known as hypervectors, which are capable of encoding, storing, and manipulating information in a manner analogous to distributed representations in the human brain. The efficiency of the technique lies in the fact that vector operations are highly parallelizable using, for example, GPUs (Grid Processing Units) or accelerators, and can be executed in real time on a very large data set. These characteristics make it particularly attractive for resource-constrained environments such as edge devices in IoT applications.

The main contribution of this work is twofold: 1) the theoretical foundation of an HDC system implementing a complete processing pipeline for anomaly classification, specifically adapted to the diverse operational scenarios encountered in industrial computing; and 2) the validation of a case study focused on anomaly detection in the behavior of emergency lighting devices. The paper is organized as follows: in the next Section we review the related works, then in Section 3 we present the system model for anomaly detection based on HDC. Section 4 describes the real-world dataset used in our experiments and in Section 5 we present the results of the evaluation performed of the proposed HDC system. Finally, in Section 6 we draw the main conclusions and suggestions for future research.

## 2 RELATED WORKS

Hyperdimensional Computing (HDC) (Kanerva, 2009) is a computational paradigm inspired by the way the human brain represents and processes information. It relies on high-dimensional mathematical structures known as hypervectors, which typically consist of thousands of components. These hypervectors act as the core units for encoding knowledge and relationships between concepts. By

means of vectorial operations, such as binding ( $\otimes$ ), bundling ( $\oplus$ ), and permutation ( $\Pi$ ) (Yu et al., 2022) (also referred as MAP, multiplication, adding, and permutation), hypervectors can be combined and transformed to represent complex data structures and abstractions.

The processing pipeline of an HDC-based classification system usually consists of three main stages: encoding, training, and classification (Chang et al., 2023; Morris et al., 2019). In the first stage, the input data space is projected into a high-dimensional space using a suitable encoding method. This encoding process is highly dependent on the nature of the input data and plays a critical role in determining the overall classification accuracy (Aygun et al., 2023). The training step aggregates all hypervectors associated to the samples in some category into a single representative hypervector, known as the class or prototypical hypervector, which will be further used in the classification process. Training may be done in a single-step, which means that each hypervector is learnt only one time or using iterative training, also called retraining (Shahhosseini et al., 2022). In the final step, classification (or inference), a test sample is encoded into a hypervector using the same encoding function and then compared against all class hypervectors to identify the class with the highest similarity.

HDC has already demonstrated its potential for anomaly detection. For instance, the work presented in (Wang et al., 2022) introduces the ODHD framework, which adopts a Positive-Unlabeled (P-U) learning strategy. In this approach, the model is trained exclusively on inlier samples, while the testing phase involves a dataset that may contain both inliers and outliers, with no label information provided. The methodology closely aligns with the approach followed in this paper. Experimental results indicate that ODHD outperforms three baseline methods (OCSVM, Isolation Forest, and Autoencoder) across six datasets and three performance metrics: accuracy, F1-score, and ROC-AUC. HDC is employed in (Arliss et al., 2024) for anomaly detection based on variations in network topology. The topology is initially represented as a time-ordered sequence of graphs, where each graph is directed and count-weighted. Subsequently, both nodes and edges are encoded as hypervectors. To identify anomalies, the method first calculates a primitive anomaly score, followed by an anomaly confidence measure obtained by referencing the score against a historical distribution of past scores. Experimental results demonstrate that this approach achieves high accuracy and efficiency on both synthetic and real-world datasets. The work presented in (Wang et al.,

2021) proposes HDAD, which leverages HDC to validate sensor readings collected by autonomous vehicles prior to their use in downstream decision-making. In this approach, sensor data is first encoded into an intermediate hypervector, which is then decoded into a reconstructed feature vector. The system compares the reconstructed vector with the original test sample to assess its validity. The authors report achieving 100% detection accuracy on a real-world dataset of vehicle sensor readings. In the IoT scope, HDC has been used for the detection of network attacks and evaluated on the NSL-KDD dataset in (Ghajari et al., 2025). The model achieved an accuracy of 91.55% on the KDDTrain+ subset, outperforming traditional approaches. The work presented in (Shahhosseini et al., 2022) introduces a method based on Hyperdimensional Computing (HDC) for learning end-user physiological and behavioral patterns, specifically targeting health-related applications on wearable devices. The proposed approach employs iterative learning to refine the model over time, achieving up to 35.8× faster training performance compared to state-of-the-art deep neural network (DNN) algorithms, while maintaining comparable accuracy.

### 3 SYSTEM MODEL

We consider a set of  $n$  devices  $d_1, d_2, \dots, d_n$  tailored to monitor a wide variety of physical and environmental phenomena of interest, including, for instance, motion, vibration, or gas concentration. In order to perform this task, such devices are carefully designed and equipped with custom hardware (e.g. MCU, advanced sensors, battery) and are provided with communication capabilities to transmit periodically data of interest, as well as their status and self-diagnostic information, towards a concentrator node  $d_0$ , also known as the sink, which stores the data and subsequently forwards it to an external network. Along the path from the transmitting device towards the sink, each device selects the next neighbor for data communication by considering several criteria, e.g. the quality of the link, thus creating a dynamic multi-hop mesh network with the sink as the final destination within the local network.

During each transmission period, every device  $d_i, 1 \leq i \leq n$  generates a text message by aggregating the values of a set of  $k$  characteristics, that is, the variables of interest. These features provide information about the device's identity, status (e.g., battery level), or network diagnostics (e.g., next hop, trip time, link quality), among others. Most of the features collected by the devices are non-deterministic,

which means that their values can fluctuate within a broad range that may still be considered normal for a given device. However, device failures can still occur, which may result in the transmission of outliers that may take one or more features and that significantly differ from the values that are considered normal.

In practice, data transmitted over time by devices are aggregated into a file of features or dataset. Let  $\mathcal{R}$  be the data space composed of all data transmitted by the  $n$  devices during a period of time and let  $\mathcal{R}_i$  and  $\mathcal{R}_{i,j}$  be the subsets of  $\mathcal{R}$  that contain all values for device  $d_i$  and all values that take the feature  $j$  of device  $d_i$ , respectively, with  $1 \leq j \leq k$  and  $1 \leq i \leq n$ . Given a set of samples  $\mathcal{R}_{i,j}$  corresponding to device  $d_i$  and feature  $j$ , we define its training set  $\mathcal{R}_{i,j}^\uparrow$  as the ascendingly sorted set of samples, i.e.  $\mathcal{R}_{i,j}^\uparrow = \{r_{i,j}^{\min}, \dots, r_{i,j}^{\max}\}$ . A single data point  $r_{i,j}^x \in \mathcal{R}_{i,j}^\uparrow$  represents the  $x^{th}$  sample taken by feature  $j$  of device  $d_i$ . The sequence of the  $x^{th}$  data points of all features  $j$  is denoted as  $r_i^x = \{r_{i,0}^x, r_{i,1}^x, \dots, r_{i,k-1}^x\}$ . We denote as  $r_{i,j}^{\min}$  and  $r_{i,j}^{\max}$  to the minimum and maximum value, respectively, in the subset  $\mathcal{R}_{i,j}^\uparrow$ , with  $r_{i,j}^{\max} - r_{i,j}^{\min} > 0$ .

#### 3.1 Encoding

We define an encoding function  $\phi: \mathcal{R} \rightarrow \mathcal{H}$  that maps each data point  $r_{i,j}^x$  in  $\mathcal{R}$  into a hypervector  $h \in \mathcal{H}$ , with  $h = \{-1, 1\}^D$  and  $D$  is the dimension of the hypervectors (generally, greater or equal than 10,000). The encoding function  $\phi$  must be carefully designed to generate hypervectors that preserve the correlation structure of the input data. In this work,  $\phi$  is based on a quantization-based approach as described in (Basaklar et al., 2021), that discretizes the input space to enable consistent and meaningful hypervector representations.

Let  $M$  be the number of quantization levels and let  $Q$  be the set of level hypervectors, i.e.  $Q = \{q_0, q_1, \dots, q_{M-1}\}$ , where  $q_m$  is the  $m^{th}$  level hypervector. The first level hypervector  $q_0$  is initialized randomly; the rest of level hypervectors, i.e.,  $\{q_1, \dots, q_{M-1}\}$ , are generated incrementally from the previous one, specifically,  $q_{i+1}$  differs from its predecessor  $q_i$  by flipping exactly  $b = \frac{D}{2(M-1)}$  components, which ensures a uniform and gradual increase in dissimilarity across levels. Note that, in doing so, we produce a matrix of  $M \times D$  and a three-dimensional tensor of  $k \times M$  level hypervectors to represent the level hypervectors of each  $d_i$ .

Let us define the function  $\mathcal{L}: \mathcal{R} \rightarrow Q$  that returns the level hypervector corresponding to a single data point, that is defined as follows:

$$\mathcal{L}(r_{i,j}^x) = \begin{cases} q_0 & x = \min \\ q_m & \min + \sigma m \leq x \leq \min + \sigma(m+1), \\ & \forall x \in (\min, \max), m \in [0, M-1) \\ q_{M-1} & x = \max \end{cases}$$

where  $\sigma$  is defined as the number of items per level, i.e.,  $\sigma = r_{i,j}^{\max} - r_{i,j}^{\min} / M$ . For each sequence of samples  $r_i^x$  in the data set, the encoding function  $\phi$  produces its corresponding hypervector, denoted as  $h_i^x$ , by adding up their level hypervectors:

$$h_i^x = \mathcal{L}(r_{i,0}^x) \oplus \mathcal{L}(r_{i,1}^x) \oplus \dots \oplus \mathcal{L}(r_{i,(k-1)}^x) \quad (1)$$

The sum operation corresponds to the bundling function  $\oplus$ , which ensures that the resulting hypervector is maximally similar to their operands, i.e. it acts by aggregating the hypervectors taken as input.

### 3.2 Training

During the training phase, the input data space is used to generate a class hypervector, also known as the prototypical hypervector, for each device  $d_i$ , denoted as  $\mathcal{H}\mathcal{V}_i$ , which is computed as:

$$\mathcal{H}\mathcal{V}_i = \bigoplus_{x \in \mathcal{R}_i} h_i^x \quad \forall x \in \mathcal{R}_i \quad (2)$$

$\mathcal{H}\mathcal{V}_i$  is constructed by applying the bundling operation to the hypervectors of all samples in the data space associated with device  $d_i$ . This process encodes the device's normal operating behavior into a reference hypervector, which is later used in the classification stage to detect deviations indicative of anomalies.

### 3.3 Classification

Let  $\{\mathcal{H}\mathcal{V}_1, \mathcal{H}\mathcal{V}_2, \dots, \mathcal{H}\mathcal{V}_n\}$  be the set of class hypervectors for  $n$  devices. Let  $t_i$  be a sequence of samples  $t_i = \{t_{i,0}, t_{i,1}, \dots, t_{i,(k-1)}\}$  to be tested. We classify the sequence  $t_i$  as an outlier of  $d_i$  iff the cosine similarity function between its hypervector  $h_i$  and the class hypervector  $\mathcal{H}\mathcal{V}_i$ , computed as:

$$\cos(h_i, \mathcal{H}\mathcal{V}_i) = \frac{h_i \times \mathcal{H}\mathcal{V}_i}{\|h_i\| \times \|\mathcal{H}\mathcal{V}_i\|} \quad (3)$$

is below a certain threshold  $S_i^+$ . It is worth noting that the smaller the difference between  $h_i$  and  $\mathcal{H}\mathcal{V}_i$ , the higher the resulting cosine similarity. Additionally, the choice of the threshold value  $S_i^+$  is critical to system performance, as it directly influences classification precision.

## 4 REAL-WORLD DATASET DESCRIPTION

The HDC system proposed has been evaluated using a case study focused on anomaly detection in emer-

gency lighting devices. To this purpose, we consider a system of  $n$  emergency lighting devices distributed in one or several nearby buildings. These devices are intended to activate automatically when the main power supply fails (e.g. during outages or blackouts), ensure visibility and safety in enclosed spaces by providing essential lighting, evacuation routes, and access to emergency exits. To ensure that devices function properly, hyperdimensional computation was used to learn from the behavior that these devices have and, with this information, to detect anomalies that may occur, thus indicating the malfunction of one or several emergency lighting devices.

The emergency lighting devices send two types of messages at different frequencies: 1) *status*, which contains the data related to the device state and 2) *diagnosis*, which contains the device diagnostic data. During two periods of 6 and 24 days respectively, from December 7th to December 12th and from December 24th to January 16th, we collect both types of data (status and diagnosis) from the  $n$  devices, which were merged into one single dataset, where a sample is formed by the combination of one sample from the status data and one other sample from the diagnostic data for a single device. The total number of samples collected thus is 1820172, each with 117 variables, resulting in a dataset of size 4.7GB. Since the number of total variables in each sample was very high, we implemented an algorithm to discard those variables whose values were poorly correlated with other variables and variables providing a low value. After applying this filtering, we obtained a final real-world dataset with the characteristics described in Table 1. Figure 1 shows the correlation matrix of  $k \times k$  variables and for an emergency lighting device, where red and blue indicate proportional and inversely proportional correlation, respectively, while gray indicates that the variables are not related.

Table 1: Real-world dataset parameters.

| Param.  | Description                      | Value   |
|---------|----------------------------------|---------|
| $S$     | Number of samples in the dataset | 1820172 |
| $V$     | Number of variables discarded    | 81      |
| $W$     | Size of the real-world dataset   | 257 MB  |
| $p$     | % of samples used for training   | 80      |
| $1 - p$ | % of samples used for testing    | 20      |

### 4.1 Threshold Selection

The resulting dataset was subsequently divided into two different subsets. The first subset, comprising  $p=80\%$  of the total samples, was used to train and encode the prototypical hypervectors that character-

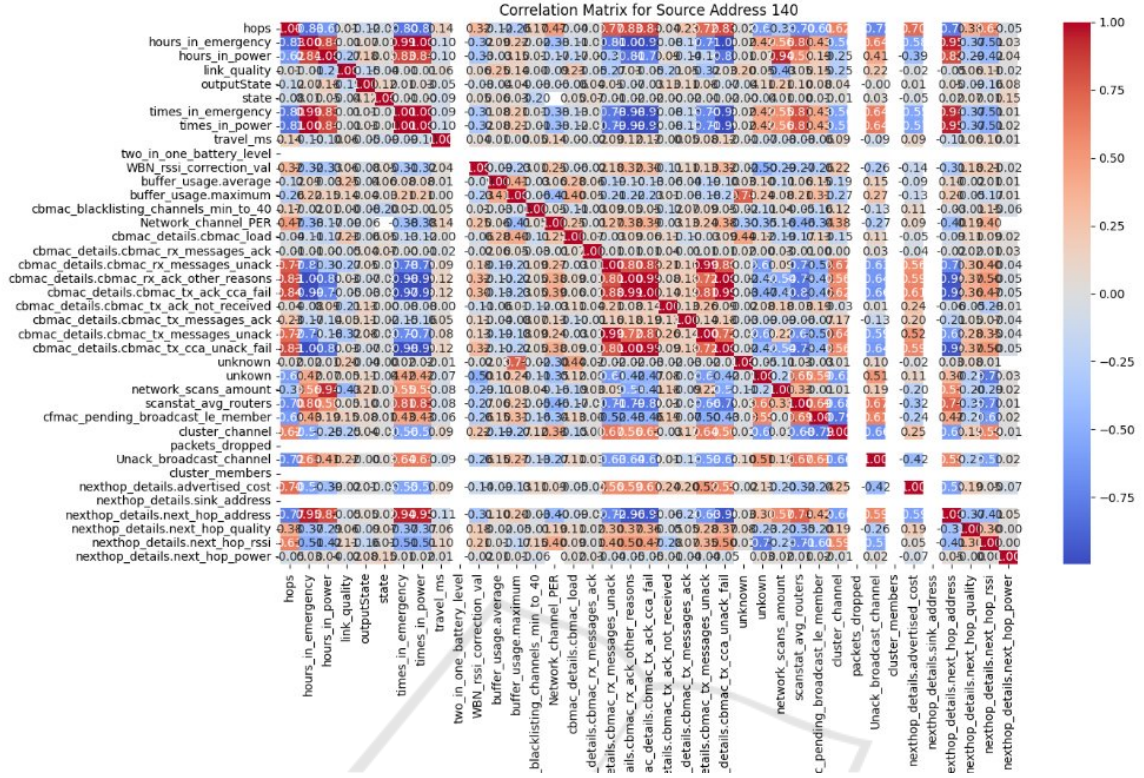


Figure 1: Correlation matrix obtained for an emergency lighting device.

ize the normal behavior of the devices, as described in Section 3. The second subset, comprising  $1 - p$  of the total samples, was used to test the HDC model. We assume that each sample in the training dataset is considered a normal behavior of an emergency lighting device so, consequently, the definition of anomaly is based on the similarity between the hypervector of the test sample and the prototypical hypervector by using the similarity cosine function, as formulated in Eq 3. For the sake of simplicity we have defined the same thresholds for all emergency devices with the next values:  $S^+ = \{0.8, 0.85, 0.9, 0.95\}$ .

## 5 EXPERIMENTATION

We have fully implemented the HDC model explained in Section 3 using the TorchHD Python library (Heddes et al., 2023), and we have executed it on a laptop with an 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz CPU and 16 GB RAM, and a NVIDIA RTX 3050 4GB GDDR6.

Table 2 summarizes the parameters used in the implementation. In Listing 1 we show the Python code to generate  $M$  level hypervectors for a single feature, based on the quantization-based approach, as detailed

Table 2: Summary of parameters used in the experiments.

| Param. | Description                       | Value    |
|--------|-----------------------------------|----------|
| $n$    | Number of devices                 | 118      |
|        | Number of hypervector classes     |          |
| $k$    | Number of variables               | 36       |
| $D$    | Dimension of the hypervectors     | 10000    |
| $M$    | Number of level hypervectors (LH) | [1,3336] |
| $b$    | Changes between consecutive LH    | [1,2500] |

in Section 3.1. This code was parallelized on a GPU to optimize its performance.

In line 3, the first level hypervector is randomly generated. Line 6 computes the number of changes  $b$  between consecutive level hypervectors as a function of  $D$  and  $M$ . In lines from 14 to 20 a set of indices is created in order to avoid repetitions of components flipped from one hypervector to another. In line 21 we flip the selected components from the previous level hypervector. In line 22 the new hypervector is saved so it can be used in the following iteration. This procedure will be performed  $M$  times resulting in a matrix of  $M \times D$ .

We have evaluated the model using different  $M$  numbers of level hypervectors, which are calculated by varying the range of items that flip between two consecutive hypervectors  $b$  from 1 to 2500. Note that

```

1 def range_hdv_levels(D, M):
2     matrix = []
3     fst_hdv = torchhd.random(1, D,
4                               "MAP", dtype=torch.int8)
5     matrix.append(fst_hdv[0])
6     b = round(D / (2 * (M - 1)))
7     if b == 0: b = 1
8     idx = set()
9     for i in range(M):
10        if i == 0:
11            prev_tsr = matrix[0].clone()
12        else:
13            prev_tsr = matrix[i-1].clone()
14            idx_av = list(set(range(D)) - idx)
15            if len(idx_av) < b:
16                idx = set()
17                idx_av = list(range(D))
18            selected = random.sample(idx_av, b)
19            idx.update(selected)
20            indices = torch.tensor(selected)
21            prev_tsr[indices] *= -1
22            matrix[i] = prev_tsr
23     return matrix
    
```

Listing 1: Generation of a matrix of correlated hypervectors.

the number of level hypervectors is an important metric to consider, as it poses an interesting tradeoff between model accuracy and resources employed. We have computed  $M$  as a function of  $b$ , and the results can be seen in Figure 2.

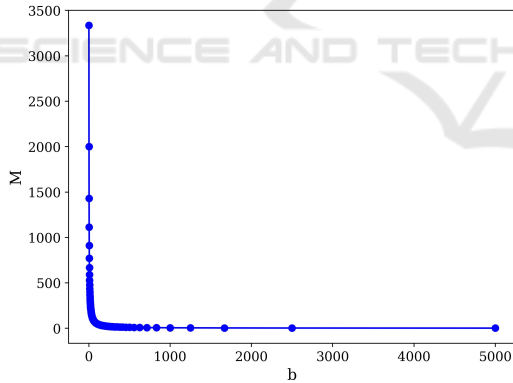


Figure 2: Number of  $M$  for each  $b$ .

## 5.1 Accuracy

The objective of the system is to determine whether the similarity between two hypervectors, one representing the  $x^{th}$  sample and other a hypervector of reference of the device that represents its normality, is lower than a certain threshold such that  $x^{th}$  is classified as an anomaly. Specifically, we are interested in evaluating the accuracy of our HDC model, by accounting the number of samples correctly classified with respect to the total number of samples.

The first test was conducted using the testing dataset, where each sample is assumed to represent normal behavior. Figure 3 illustrates the accuracy of the HDC model across different threshold values for  $S^+$  and for different values of  $b$ . As expected, increasing the threshold results in decreased accuracy. This behavior arises because achieving a high cosine similarity value requires the test sample to closely match the reference. Consequently, the best results were obtained with a threshold of 0.8. In contrast, when the threshold was increased to 0.95, the model's accuracy dropped to nearly zero, which means that the majority of test samples fail to meet the stricter threshold. On the other hand, the highest accuracy is obtained with  $b = 1$ ; note that, however, for small values of  $b$ , the accuracy of the system remains similar to the results obtained with  $b = 1$ .

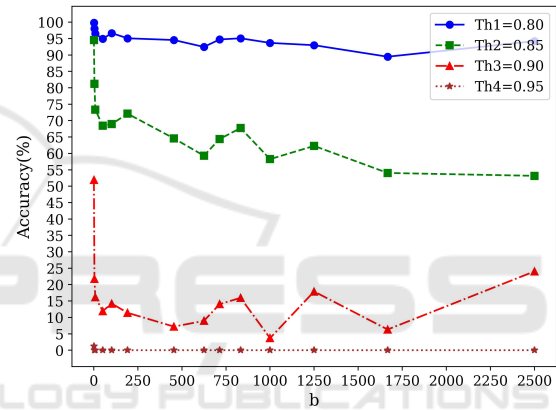


Figure 3: Accuracy of the model for different values of  $b$ .

In order to evaluate how the HDC model behaves with samples whose values differ significantly from the values in the dataset that, as explained, represent the normal behavior of the device, we had to generate synthetic data. The synthetic data was generated by repeatedly selecting the first 100 samples from the dataset and applying two different deviation values: 0.5 and 0.9. For each case, the same set of 100 entries was used: first, the deviation was applied to a single characteristic; then, to two characteristics; and progressively to more, until all characteristics were included. This procedure resulted in the generation of two more datasets that were used to test the performance of the system anomaly detection. Both datasets have a total of 3400 samples each and a total size of 3.9MB. The results of testing the HDC system with the synthetic data may be observed in Figure 4. The left side of the figure shows the accuracy for a deviation of 0.5, while the right side corresponds to a deviation of 0.9. As observed, when the threshold is set to 0.95 or 0.9, the accuracy remains close to 100%

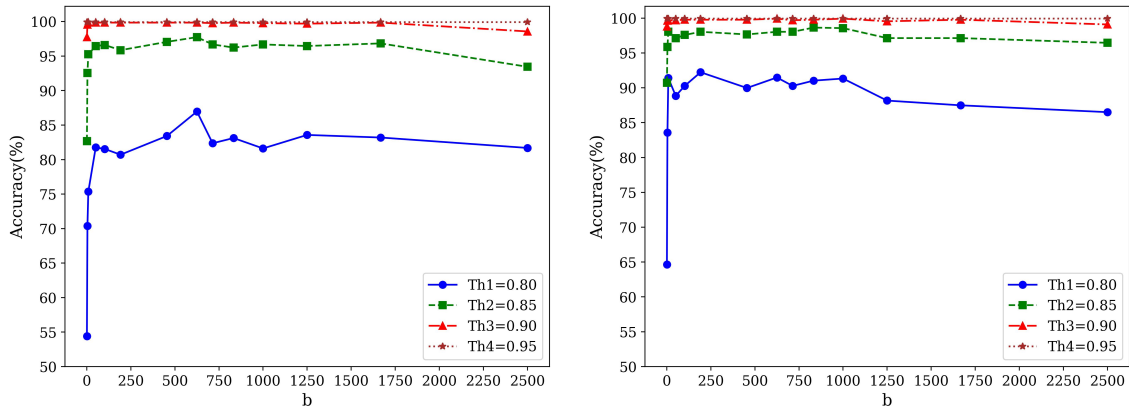


Figure 4: Accuracy of the model with deviations on the synthetic data of 0.5 (left) and 0.9 (right) for different thresholds.

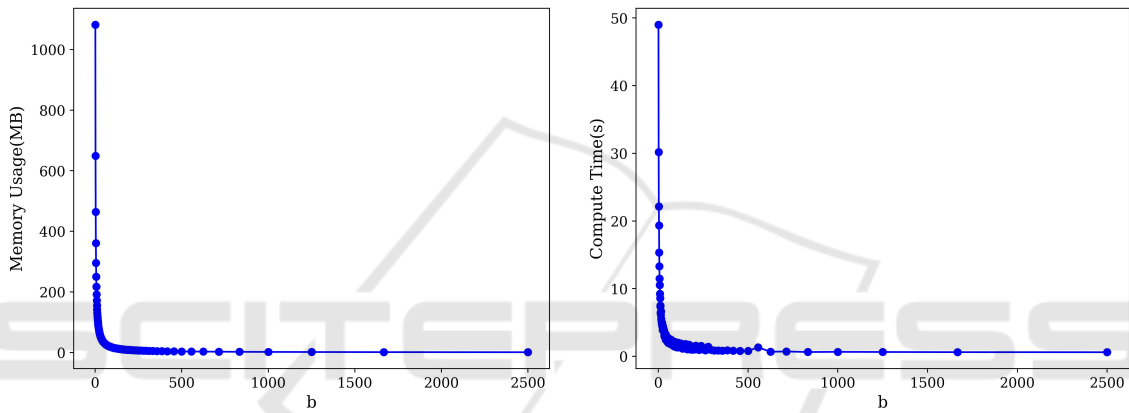


Figure 5: Memory RAM consumption (left) and execution time (right) for different values of  $b$ .

across all values of  $b$ . In contrast, lower threshold values result in a noticeable decrease in model accuracy. These results are due to the fact that these synthetic data samples are expected to be classified as anomalies. Therefore, with a higher threshold, it is more likely that the cosine similarity of a given sample does not reach the threshold, resulting in its classification as an anomaly and, consequently, in high accuracy. In contrast, with a lower threshold, it is more likely that samples reach the threshold, leading to misclassifications and, therefore, lower accuracy.

## 5.2 Memory Usage and Execution Times

HDC systems are influenced by several key parameters that critically affect the resource-performance tradeoff, as outlined in Table 2. Specifically, the hypervector dimensionality ( $D$ ), the number of input features ( $k$ ), and the total number of level hypervectors ( $M$ ) have a substantial impact on RAM consumption, with the storage cost scaling as  $O(k \times M \times D)$ . In

addition, the size of the dataset used for training and testing plays a critical role in system design and evaluation. Although larger datasets can improve model accuracy, they also impose greater demands on memory, computational power, and processing time, which could limit the feasibility of implementing the HDC system in an embedded system.

To illustrate this, we have computed the requirements of RAM for different values of  $b$  of our HDC system, which will store a different number of level hypervectors  $M$  for different values of  $b$ , thus affecting the RAM consumption. In Figure 5 on the left side is shown how RAM decreases when  $b$  increases, since the HDC system will require less number of level hypervectors. Figure 5 on the right side shows the impact of varying  $b$  on the time invested in computing the level hypervectors. Recall that the procedure of computing the level hypervectors was parallelized on a GPU. As observed, in the worst-case scenario with the lowest value of  $b$ , computing the corresponding  $M = 3336$  level hypervectors takes approximately 50 seconds. However, as illustrated in Figure 3, reducing

the number of level hypervectors ( $M$ ) still yields accuracy results comparable to those obtained with the maximum  $M$ . This reduction significantly enhances the feasibility of implementing the HDC system on embedded platforms.

## 6 CONCLUSIONS

We have proposed an efficient and scalable HDC system for anomaly classification in industrial environments. The system was evaluated in a real-world scenario involving 118 devices periodically transmitting status information. Our HDC system operates autonomously, without requiring prior knowledge of what constitutes an anomaly or labeled training data. As a result, the system behaves in an unsupervised manner, continuously learning normal patterns of behavior and identifying deviations that may indicate faults or abnormal conditions, making it particularly suitable for dynamic and data-scarce industrial settings. As future work, we propose the implementation of our HDC system on customized accelerators for the optimization of vector instructions.

## ACKNOWLEDGEMENTS

This paper is partially supported by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 under Grant TALENT-BELIEF (PID2020-116417RB-C44), the project OASIS (PID2023-148285OB-C41) and the Project MIRATAR TED2021-132149B-C41 funded by MCIN/AEI/10.13039/501100011033 and by European Union NextGenerationEU/PRTR.

## REFERENCES

- Arliss, W., Godbehere, A. B., and Mueller, G. (2024). Using hypervectors for efficient anomaly detection in graph streams. In *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10.
- Aygun, S., Shoushtari Moghadam, M., Najafi, M. H., and Imani, M. (2023). Learning from hypervectors: A survey on hypervector encoding.
- Baddar, S. W. A.-H., Merlo, A., and Migliardi, M. (2014). Anomaly detection in computer networks: A state-of-the-art review. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 5:29–64.
- Basaklar, T., Tuncel, Y., Narayana, S. Y., Gumussoy, S., and Ogras, U. Y. (2021). Hypervector design for efficient hyperdimensional computing on edge devices.
- Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336.
- Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3).
- Chang, C.-Y., Chuang, Y.-C., Huang, C.-T., and Wu, A.-Y. (2023). Recent progress and development of hyperdimensional computing (hdc) for edge intelligence. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):119–136.
- Ghajari, G., Ghimire, A., Ghajari, E., and Amsaad, F. (2025). Network anomaly detection for iot using hyperdimensional computing on nsl-kdd.
- Heddes, M., Nunes, I., Vergés, P., Kleyko, D., Abraham, D., Givargis, T., Nicolau, A., and Veidenbaum, A. (2023). Torchhd: an open source python library to support research on hyperdimensional computing and vector symbolic architectures. *J. Mach. Learn. Res.*, 24(1).
- Hernández-Cano, A., Matsumoto, N., Ping, E., and Imani, M. (2021). Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 56–61.
- Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1:139–159.
- Morris, J., Imani, M., Bosch, S., Thomas, A., Shu, H., and Rosing, T. (2019). Comphd: Efficient hyperdimensional computing using model compression. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6.
- Nassif, A. B., Talib, M. A., Nasir, Q., and Dakalbab, F. M. (2021). Machine learning for anomaly detection: A systematic review. *IEEE Access*, 9:78658–78700.
- Shahhosseini, S., Ni, Y., Kasaeyan Naeini, E., Imani, M., Rahmani, A. M., and Dutt, N. (2022). Flexible and personalized learning for wearable health applications using hyperdimensional computing. In *Proceedings of the Great Lakes Symposium on VLSI 2022, GLSVLSI '22*, page 357–360, New York, NY, USA. Association for Computing Machinery.
- Wang, R., Jiao, X., and Hu, X. S. (2022). Odhd: one-class brain-inspired hyperdimensional computing for outlier detection. In *Proceedings of the 59th ACM/IEEE Design Automation Conference, DAC '22*, page 43–48, New York, NY, USA. Association for Computing Machinery.
- Wang, R., Kong, F., Sudler, H., and Jiao, X. (2021). Brief industry paper: Hdad: Hyperdimensional computing-based anomaly detection for automotive sensor attacks. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 461–464.
- Yu, T., Zhang, Y., Zhang, Z., and Sa, C. D. (2022). Understanding hyperdimensional computing for parallel single-pass learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.