

# DOMINO: Domain-Invariant Hyperdimensional Classification for Multi-Sensor Time Series Data

Junyao Wang<sup>†</sup>, Luke Chen<sup>§</sup>, Mohammad Abdullah Al Faruque<sup>†§</sup>

<sup>†</sup> Department of Computer Science, University of California, Irvine, CA, United States

<sup>§</sup> Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, United States  
{junyaow4, panwangc, alfaruqu}@uci.edu

**Abstract**—With the rapid evolution of the Internet of Things, many real-world applications utilize heterogeneously connected sensors to capture time-series information. Edge-based machine learning (ML) methodologies are often employed to analyze locally collected data. However, a fundamental issue across data-driven ML approaches is *distribution shift*. It occurs when a model is deployed on a data distribution different from what it was trained on, and can substantially degrade model performance. Additionally, increasingly sophisticated deep neural networks (DNNs) have been proposed to capture spatial and temporal dependencies in multi-sensor time series data, requiring intensive computational resources beyond the capacity of today's edge devices. While brain-inspired hyperdimensional computing (HDC) has been introduced as a lightweight solution for edge-based learning, existing HDCs are also vulnerable to the distribution shift challenge. In this paper, we propose DOMINO, a novel HDC learning framework addressing the distribution shift problem in noisy multi-sensor time-series data. DOMINO leverages efficient and parallel matrix operations on high-dimensional space to dynamically identify and filter out domain-variant dimensions. Our evaluation on a wide range of multi-sensor time series classification tasks shows that DOMINO achieves on average 2.04% higher accuracy than state-of-the-art (SOTA) DNN-based domain generalization techniques, and delivers 16.34× faster training and 2.89× faster inference. More importantly, DOMINO exhibits notably better performance when learning from partially labeled data and highly imbalanced data, and provides 10.93× higher robustness against hardware noises than SOTA DNNs.

## I. INTRODUCTION

The Internet of Things (IoT) has become an emerging trend for its extraordinary potential to connect heterogeneous devices and enable them with new capabilities [1]. Many real-world IoT applications utilize multiple sensors to collect information over the course of time, constituting multi-sensor time series data [2]–[4]. These applications often leverage edge-based machine learning (ML) algorithms to analyze locally collected data and perform various learning tasks. However, a critical issue across data-driven ML approaches, including deep neural networks (DNNs), is *distribution shift*. In particular, the excellent performance of these ML algorithms relies heavily on the critical assumption that the training and inference data are independently and identically distributed (i.i.d.); i.e., they come from the same distribution [5]. Unfortunately, this assumption can be easily violated in real-world scenarios and have shown to substantially degrade model performance in many embedded ML applications, where instances from unseen domains not fitting the distribution of the training data are inevitable [6]–[8]. For instance, in the field of mobile health,

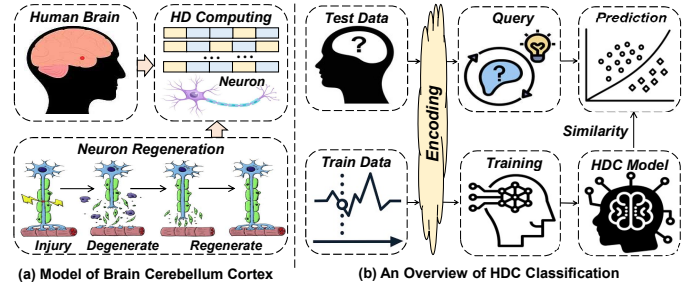


Fig. 1. An Overview of Brain Cerebellum Cortex and HDC Classification

models can systematically fail when tested on patients from different hospitals or people from diverse demographics [9].

A number of innovative *domain generalization* (DG) techniques have been proposed for deep learning (DL) [10], [11]. However, due to their weak notion of memorization, these DL approaches often fail to perform well on noisy multi-sensor time series data with spatial and temporal dependencies [12]. Recurrent neural networks (RNNs), e.g., long short-term memory (LSTM), have recently been proposed to address this issue [13]–[15]. Nevertheless, these models are notably complicated and inefficient to train, and their intricate architectures require substantial off-chip memory and computational power to iteratively refine millions of parameters over multiple time periods [15]. Such resource-intensive requirements can be impractical for less powerful computing platforms. Considering the massive amount of information nowadays, the power and memory limitations of embedded devices, and the potential instabilities of IoT systems, a more lightweight, efficient, and scalable learning framework to combat the distribution shift issue in multi-sensor time series data are of critical need.

In contrast to traditional AI methodologies, brain-inspired Hyperdimensional Computing (HDC) incorporates learning capability along with typical memory functions of storing/loading information, and hence brings unique advantages in dealing with time-series data [16]. Additionally, HDC provides a powerful learning solution for today's edge platforms by providing notably fast convergence, high computational efficiency, ultra-robustness against noises, and lightweight hardware implementation [17]–[19]. As demonstrated in Fig. 1(a), HDC is motivated by the neuroscience observation that the cerebellum cortex in human brains effortlessly and efficiently process memory, perception, and cognition information without much concern for noisy or broken neuron cells. Closely mimicking the information representation and functionalities of human brains, HDC *encodes* low-dimensional inputs to

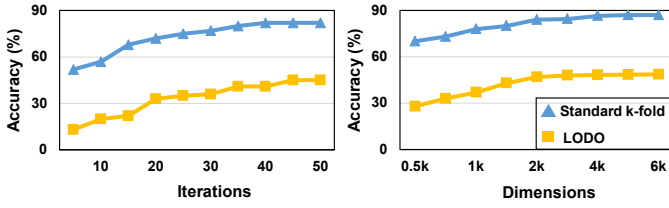


Fig. 2. Comparison of the Accuracy of LODO CV and Standard  $k$ -fold CV

*hypervectors* with thousands of elements to perform various learning tasks [19] as shown in Fig. 1(b). HDC then conducts highly parallel and well-trackable operations, and has been proven to achieve high-quality results in various learning tasks with comparable accuracy to SOTA DNNs [18]–[20].

Unfortunately, existing HDCs are not immune to the distribution shift issue. As shown in Fig. 2, SOTA HDCs converge at notably lower accuracy in leave-one-domain-out (LODO) cross-validation (CV) than in standard  $k$ -fold CV regardless of training iteration and model complexity. LODO CV involves training a model on all the available data except for one domain that is left out for inference, while standard  $k$ -fold CV randomly divides all data into  $k$  subsets with  $k - 1$  subsets for training and the remaining one for inference. Such performance degradation indicates a very limited generalization capability of existing HDCs. However, standard  $k$ -fold CV does not reflect the real-world distribution shift problem, as the random sampling process introduces *data leakage* that enables the training data to include information from all the domains. The accuracy from the standard  $k$ -fold CV is thus often considerably higher than that in real-world scenarios. In contrast, with constant dynamic regeneration of cerebral neurons shown in Fig. 1(a), humans are capable of effectively identifying and recalling attributes shared across different domains, and naturally filtering out information that is too specific and biased towards a single domain. Humans then utilize this shared knowledge to infer instances from novel domains. While the goal of HDC is to exploit the high-dimensionality of randomly generated vectors to represent information mimicking a pattern of neural activity, it remains challenging for existing HDCs to support a similar behavior.

In this paper, we propose DOMINO, a novel HDC domain generalization (DG) algorithm for multi-sensor time series classification. DOMINO effectively eliminates dimensions representing domain-variant information to enhance model generalization capability. Our main contributions are listed below:

- To the best of our knowledge, DOMINO is the first HDC-based DG algorithm. By dynamically identifying and regenerating domain-variant dimensions, DOMINO achieves on average 2.04% higher accuracy than SOTA DL-based techniques with  $16.34\times$  faster training and  $2.89\times$  faster inference, ensuring accurate and timely performance for DG.
- DOMINO achieves up to 5.81% higher accuracy than SOTA DNNs in scenarios with a significantly limited amount of labeled data and on average 2.58% higher accuracy when learning from highly imbalanced data across different domains. Additionally, DOMINO exhibits  $10.93\times$  higher robustness against hardware noise than SOTA DNNs.

- We propose hardware-aware optimizations for DOMINO's implementation, and evaluated it across multiple embedded hardware devices including Raspberry Pi and NVIDIA Jetson Nano. DOMINO exhibits considerably lower inference latency and energy consumption than DL-based approaches.

## II. METHODOLOGY

The goal of DOMINO is to effectively leverage the information learned from each training iteration to identify and filter out domain-variant dimensions, thereby enhancing the generalization capability of our model. As shown in Fig. 3, DOMINO starts with encoding multi-sensor time series data samples into high-dimensional space (A). DOMINO then conducts two innovative steps, *domain-specific modeling* and *domain generalization*, to enable its encoding module and randomly-generated base vectors with awareness of the relevance of each dimension to the domains. In each iteration of *domain-specific modeling*, DOMINO exploits an efficient and parallel hyperdimensional learning algorithm to construct domain-specific hyperdimensional models (D). In *domain generalization*, we utilize domain-specific models to form class-specific matrices (E), and then identify and filter out dimensions that represent domain-variant information (F). To mitigate the performance degradation caused by eliminating dimensions, we replace these dimensions with randomly generated hypervectors and retrain them. Note that all these operations can be done in a highly parallel matrix-wise way, as multiple training samples can be grouped into a matrix of row hypervectors.

### A. HDC Preliminaries

Inspired by the high-dimensional information representation in human brains, HDC maps inputs onto hyperdimensional space as *hypervectors* (A), each of which contains thousands of elements. One unique property of the hyperdimensional space is the existence of a large number of nearly orthogonal hypervectors, enabling highly parallel operations such as similarity calculations, bundlings, and bindings. Mathematically, consider random bipolar hypervectors  $\mathcal{H}_1$  and  $\mathcal{H}_2$  with dimension  $\mathcal{D}$ , i.e.,  $\mathcal{H}_1, \mathcal{H}_2 \in \{-1, 1\}^{\mathcal{D}}$ , when  $\mathcal{D}$  is large enough, the dot product  $\mathcal{H}_1 \cdot \mathcal{H}_2 \approx 0$ . **Similarity:** calculation of the distance between the query hypervector and the class hypervector (noted as  $\delta(\cdot, \cdot)$ ). For real-valued hypervectors, a common measure is cosine similarity. For bipolar hypervectors, it is simplified to the Hamming distance. **Bundling (+):** element-wise addition of multiple hypervectors, e.g.,  $\mathcal{H}_{bundle} = \mathcal{H}_1 + \mathcal{H}_2$ , 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,  $\delta(\mathcal{H}_{bundle}, \mathcal{H}_1) \gg 0$  while  $\delta(\mathcal{H}_{bundle}, \mathcal{H}_3) \approx 0$  ( $\mathcal{H}_3 \neq \mathcal{H}_1, \mathcal{H}_2$ ). **Binding (\*):** element-wise multiplication associating two hypervectors to create another near-orthogonal hypervector, i.e.  $\mathcal{H}_{bind} = \mathcal{H}_1 * \mathcal{H}_2$  where  $\delta(\mathcal{H}_{bind}, \mathcal{H}_1) \approx 0$  and  $\delta(\mathcal{H}_{bind}, \mathcal{H}_2) \approx 0$ . Due to reversibility, in bipolar cases,  $\mathcal{H}_{bind} * \mathcal{H}_1 = \mathcal{H}_2$ , information from both hypervectors can be preserved. Binding models how

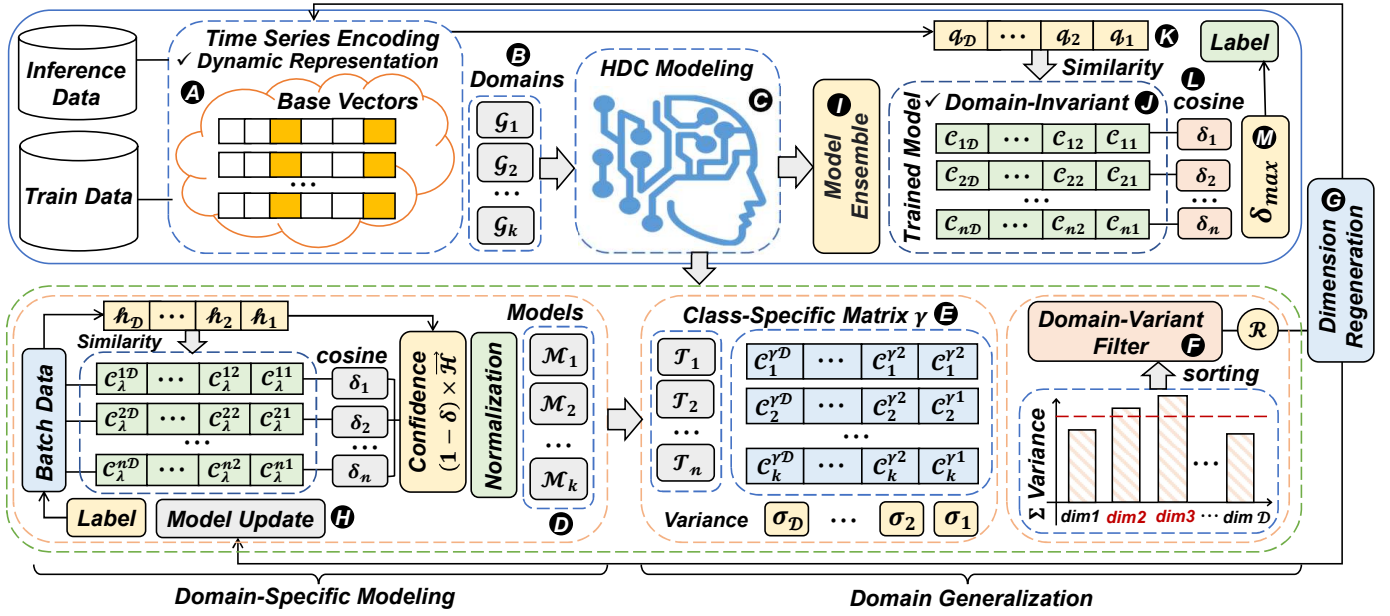


Fig. 3. An Overview of DOMINO Workflow

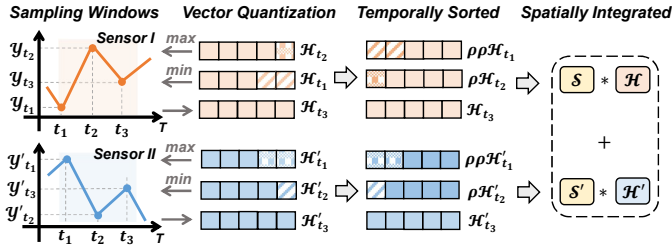


Fig. 4. HDC Encoding Technique for Multi-Sensor Time Series Data

human brains *connect* input information. **Permutation** ( $\rho$ ): a single circular shift of a hypervector by moving the value of the final dimension to the first dimension and shifting all other values to their next dimension. It generates a permuted hypervector that is nearly orthogonal to its original hypervector, i.e.,  $\delta(\rho\mathcal{H}, \mathcal{H}) \approx 0$ . Permutation models how human brains handle *sequential* information. **Inference**: The inference of HDC consists of two steps: (i) encode (A) inference data with the same encoder in training to generate a query hypervector  $\mathcal{Q}$  (K), and (ii) calculate the distance or similarity between  $\mathcal{Q}$  and each class hypervector (L).  $\mathcal{Q}$  is then classified to the class where it achieves the highest similarity (M).

### B. Encoding of Multi-Sensor Time Series

To capture the temporal and spatial dependencies in multi-sensor time series data, we employ the encoding techniques demonstrated in Fig. 4. We sample time series data in  $n$ -gram windows; in each sample window, the signal values ( $y$ -axis) store the information and the time ( $x$ -axis) represents the temporal sequence. We first assign random hypervectors  $\mathcal{H}_{max}$  and  $\mathcal{H}_{min}$  to represent the maximum and minimum signal values, i.e.,  $y_{max}$  and  $y_{min}$ . We then perform vector quantization to values between  $y_{max}$  and  $y_{min}$  to generate vectors that have a spectrum of similarity to  $\mathcal{H}_{max}$  and  $\mathcal{H}_{min}$ .

As shown in Fig. 4, Sensor I and Sensor II follow a time series in trigram. Sensor I has the maximum value at  $t_2$  and the minimum value at  $t_1$ , and hence we assign randomly generated hypervectors  $\mathcal{H}_{t_2}$  and  $\mathcal{H}_{t_1}$  to  $y_{t_2}$  and  $y_{t_1}$ . Similarly, we assign randomly generated hypervectors  $\mathcal{H}'_{t_1}$  and  $\mathcal{H}'_{t_2}$  to  $y'_{t_1}$  and  $y'_{t_2}$  in Sensor II. We then assign hypervectors to  $y_{t_3}$  in Sensor I and value at  $y'_{t_3}$  in Sensor II through vector quantization, i.e.,

$$\mathcal{H}_{t_3} = \mathcal{H}_{t_1} + \frac{y_{t_3} - y_{t_1}}{y_{t_2} - y_{t_1}} \cdot (\mathcal{H}_{t_2} - \mathcal{H}_{t_1})$$

$$\mathcal{H}'_{t_3} = \mathcal{H}'_{t_2} + \frac{y'_{t_3} - y'_{t_2}}{y'_{t_1} - y'_{t_2}} \cdot (\mathcal{H}'_{t_1} - \mathcal{H}'_{t_2}).$$

We then represent the temporal sequence of data samples utilizing permutations as explained in section II-A. In Fig. 4, for Sensor I and Sensor II, we perform rotation shift ( $\rho$ ) twice to  $\mathcal{H}_{t_1}$  and  $\mathcal{H}'_{t_1}$ , once to  $\mathcal{H}_{t_2}$  and  $\mathcal{H}'_{t_2}$ , and keep  $\mathcal{H}_{t_3}$  and  $\mathcal{H}'_{t_3}$  the same. We bind data samples in one sampling window by calculating  $\mathcal{H} = \rho\rho\mathcal{H}_{t_1} * \rho\mathcal{H}_{t_2} * \mathcal{H}_{t_3}$  and  $\mathcal{H}' = \rho\rho\mathcal{H}'_{t_1} * \rho\mathcal{H}'_{t_2} * \mathcal{H}'_{t_3}$ . Finally, to integrate data from multiple sensors, we generate a random signature hypervector for each sensor and bind information as  $\mathcal{S}_1 * \mathcal{H}_1 + \mathcal{S}_2 * \mathcal{H}_2 + \dots + \mathcal{S}_n * \mathcal{H}_n$ , where  $\mathcal{S}_i$  denote the signature hypervector for Sensor  $i$ , and  $\mathcal{H}_i$  denote the data from sensor  $i$ . In Fig. 4, we combine information from Sensor I and Sensor II by randomly generating signature hypervectors  $\mathcal{S}$  and  $\mathcal{S}'$  and calculating  $\mathcal{S} * \mathcal{H} + \mathcal{S}' * \mathcal{H}'$ .

### C. Domain-Specific Modeling

As demonstrated in Fig. 3, after mapping training data samples to high-dimensional space (A) utilizing the time-series encoding technique in section II-B, DOMINO separates training data into  $k$  subsets ( $k$  = the number of domains) based on their domains (B). We then exploit an efficient and lightweight hyperdimensional learning algorithm to generate a domain-specific model for each domain (D). Our approach



**Algorithm 1** Domain-Specific Modeling

**Input:**  $\mathcal{N}$  encoded training samples  $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_N$  with  $n$  classes and  $k$  domains, domains for training data  $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_k$ , label of each data sample  $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_N$ , learning rate  $\eta$ .

**Output:**  $k$  domain-specific models  $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k$  after one training iteration.

- 1: **for each**  $\lambda \in [1, k]$  **do**
- 2:   Initialize a domain-specific model  $\mathcal{M}_\lambda$  consisting of one class hypervectors for each class  $\mathcal{M}_\lambda = \{\mathcal{C}_\lambda^1, \mathcal{C}_\lambda^2, \dots, \mathcal{C}_\lambda^n\}$
- 3:   **for each**  $\mathcal{H}_i \in \{\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_N\}$  **do**
- 4:     **if**  $\text{domain}(\mathcal{H}_i) = \mathcal{G}_\lambda$  **then**
- 5:        $\mathcal{C}_{\max} = \arg \max_{\mathcal{C}_\lambda^t} \{\delta(\mathcal{H}_i, \mathcal{C}_\lambda^1), \dots, \delta(\mathcal{H}_i, \mathcal{C}_\lambda^n)\}$
- 6:       **if**  $\mathcal{L}_i = \mathcal{C}_{\max}$  **then**
- 7:         continue
- 8:       **else if**  $\mathcal{L}_i \neq \mathcal{C}_{\max} \wedge \mathcal{L}_i = \mathcal{C}_\lambda^j$  **then**
- 9:          $\mathcal{C}_{\max} \leftarrow \mathcal{C}_{\max} - \eta \cdot [1 - \delta(\mathcal{H}_i, \mathcal{C}_{\max})] \times \mathcal{H}_i$
- 10:         $\mathcal{C}_\lambda^j \leftarrow \mathcal{C}_\lambda^j + \eta \cdot [1 - \delta(\mathcal{H}_i, \mathcal{C}_\lambda^j)] \times \mathcal{H}_i$
- 11:    $\mathcal{M}_\lambda = \text{Normalize}(\mathcal{M}_\lambda)$
- 12: **return**  $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k$

aims to provide accurate classification performance by identifying common patterns during training and eliminating model saturations. As shown in Algorithm 1, we bundle encoded data points by scaling a proper weight to each of them depending on how much new information is added to class hypervectors. For each domain  $\mathcal{G}_\lambda$  ( $1 \leq \lambda \leq k$ ), a new encoded training sample  $\mathcal{H}$  updates the domain-specific model  $\mathcal{M}_\lambda$  based on its cosine similarities with all class hypervectors (line 5), i.e.,

$$\delta(\mathcal{H}, \mathcal{C}_\lambda^t) = \frac{\mathcal{H} \cdot \mathcal{C}_\lambda^t}{\|\mathcal{H}\| \cdot \|\mathcal{C}_\lambda^t\|} = \frac{\mathcal{H}}{\|\mathcal{H}\|} \cdot \frac{\mathcal{C}_\lambda^t}{\|\mathcal{C}_\lambda^t\|} \propto \mathcal{H} \cdot \text{Normalize}(\mathcal{C}_\lambda^t)$$

where  $\mathcal{H} \cdot \mathcal{C}_\lambda^t$  ( $1 \leq t \leq n$ ) is the dot product between  $\mathcal{H}$  and a class hypervector  $\mathcal{C}_\lambda^t$  representing class  $t$  of domain  $\mathcal{G}_\lambda$ . Here  $\|\mathcal{H}\|$  is a constant factor when comparing a query with all classes and thus can be eliminated. The cosine similarity calculation can hence be simplified to a dot product operation between  $\mathcal{H}$  and the normalized class hypervector. If the prediction matches the expected output, no update will be made to prevent overfitting. If  $\mathcal{H}$  has the highest cosine similarity with class  $\mathcal{C}_{\max}$  while its true label matches  $\mathcal{C}_\lambda^j$  (line 8), the model updates following lines 9 - 10. A large  $\delta(\mathcal{H}, \cdot)$  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 ( $1 - \delta(\mathcal{H}, \cdot) \approx 1$ ). We then normalize each dimension in every model at the end of our algorithm (line 11). Our learning algorithm provides a higher chance for non-common patterns to be properly included in the model, and effectively reduces computationally-expensive retraining iterations required to achieve reasonable accuracy.

**D. Domain Generalization**

In each training iteration, after applying the hyperdimensional learning algorithm, DOMINO utilizes partially trained models to calculate the relevance of each dimension to domains and filter-out dimensions representing domain-variant

**Algorithm 2** Domain Generalization

**Input:**  $k$  domain-specific models  $\{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k\}$  each with size  $n \times \mathcal{D}$ , regeneration rate  $\mathcal{R}$ .

**Output:** Domain-variant dimensions  $\mathcal{U}$  to drop.

- 1: Initialize  $n$  empty matrices  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ , each with size  $k \times \mathcal{D}$ .
- 2: **for each**  $\mathcal{T}_\gamma \in \{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n\}$  **do**
- 3:   **for each**  $\mathcal{M}_\lambda \in \{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k\}$  **do**
- 4:      $\mathcal{T}_\gamma[\lambda, :] = \mathcal{M}_\lambda[\gamma, :]$    ▷ Form  $n$  class-specific matrices
- 5:      $\sigma_\gamma = \text{Variance}(\mathcal{T}_\gamma, \text{columnwise})$    ▷  $\dim(\sigma_\gamma) = 1 \times \mathcal{D}$
- 6:  $\mathcal{V} = \sum_{i=1}^n \sigma_i$
- 7:  $\mathcal{U} = \text{argsort}(\mathcal{V})[\lfloor (1 - \mathcal{R}) \cdot \mathcal{D} \rfloor : \mathcal{D}]$    ▷  $\mathcal{R}$  of dimensions with largest variance
- 8: **return**  $\mathcal{U}$

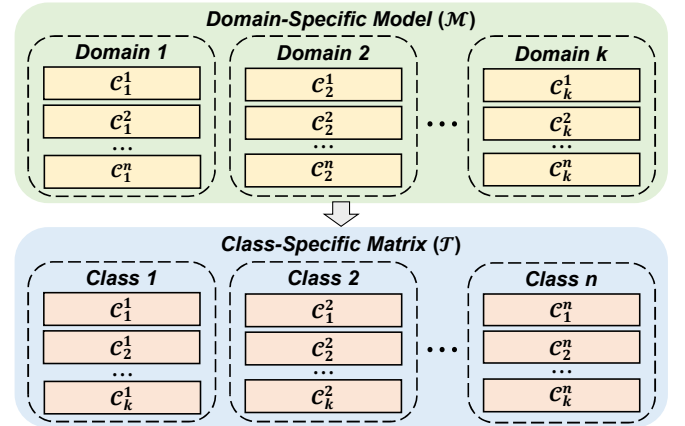


Fig. 5. Extracting Class Hypervectors From  $k$  Domain-Specific Models to Construct  $n$  Class-Specific Matrices. Each Class Hypervector is in size  $1 \times \mathcal{D}$

information. As demonstrated in Fig. 3, our *domain generalization* consists of three parts: class-specific aggregation (E), domain-variant filter (F), and model ensemble (I). In *class-specific aggregation*, we extract class hypervectors from each domain-specific model obtained in section II-C to construct class-specific matrices. In *domain-variant filter*, DOMINO identifies and regenerates domain-variant dimensions to enhance the generalization capability of our model. In *model ensemble*, we combine multiple domain-specific models into a single model to form the domain-invariant HDC model.

**Class-Specific Aggregation:** As demonstrated in Algorithm 2, for each class, we extract the class hypervector from each domain-specific model representing that class to form a class-specific matrix (line 4). As shown in Fig. 5, we denote  $\mathcal{C}_\lambda^t$  as the class hypervector of class  $t$  ( $1 \leq t \leq n$ ) in the domain-specific model  $\mathcal{M}_\lambda$  ( $1 \leq \lambda \leq k$ ), where  $k$  and  $n$  represent the number of domains and the number of classes, respectively. Then, for instance, to construct the class-specific matrix for class 1, we extract  $\mathcal{C}_1^1$  from  $\mathcal{M}_1$ ,  $\mathcal{C}_1^2$  from  $\mathcal{M}_2$ , ..., and  $\mathcal{C}_1^k$  from  $\mathcal{M}_k$ . Note that we obtain  $k$  domain-specific models each with size  $n \times \mathcal{D}$  (Algorithm 2 line 3), where  $\mathcal{D}$  denote the dimensionality of our HDC models. Hence, here we construct  $n$  class-specific matrices (line 2), each with size  $k \times \mathcal{D}$ .

**Domain-Variant Filter:** HDC algorithms represent each class with a hypervector that encodes the patterns of that class. Dimensions with very different values indicate that they store differentiated patterns, while dimensions with similar values indicate they store common information across classes. Hence,

as detailed in Algorithm 2, for each class-specific matrix, we calculate the variance of each dimension to measure its dispersion (line 5). Dimensions with large variance indicate that, for the same class, these dimensions store very differentiated information, and are hence considered domain-variant. We sum up the variance vector of each class-specific matrix to obtain  $\mathcal{V}$ , an  $1 \times \mathcal{D}$  vector representing the overall relevance of each dimension to domains (line 6). We then select the top  $\mathcal{R}$  portion of dimensions with the highest variance, where  $\mathcal{R}$  denote the regeneration rate, and filter them out from the model (line 7). To mitigate the performance degradation caused by eliminating dimensions from models, we replace them with new randomly-generated dimensions and retrain them (Fig. 3 G, H). Considering the non-linear relationship between features, we utilize an encoding method inspired by the Radial Basis Function (RBF) [21]. Mathematically, for an input vector in original space  $\mathcal{F} = \{f_1, f_2, \dots, f_n\} (f_i \in \mathbb{R})$ , we generate the corresponding hypervector  $\mathcal{H} = \{h_1, h_2, \dots, h_{\mathcal{D}}\} (0 \leq h_i \leq 1, h_i \in \mathbb{R})$  with  $\mathcal{D} (\mathcal{D} \gg n)$  dimensions by calculating a dot product of  $\mathcal{F}$  with a randomly generated vector as  $h_i = \cos(\mathcal{B}_i \cdot \mathcal{F} + c) \times \sin(\mathcal{B}_i \cdot \mathcal{F})$ , where  $\mathcal{B}_i = \{b_1, b_2, \dots, b_n\}$  is a randomly generated base vector with  $b_i \sim \text{Gaussian}(\mu = 0, \sigma = 1)$  and  $c \sim \text{Uniform}[0, 2\pi]$ . DOMINO replaces each base vector of the selected dimensions in the encoding module with another randomly generated vector from the Gaussian distribution, and retrains the current domain-specific models. Instead of training from scratch, DOMINO only updates values of class hypervectors on the dropped dimensions while other dimensions continue learning based on their existing values.

**Model Ensemble:** After filtering out and regenerating domain-variant dimensions, we assemble all the domain-specific models to build a general domain-invariant model. We construct this model based on domain-specific models and the weight of each domain. For each domain  $\lambda \in [0, k]$  ( $k$  = number of domains), we calculate the proportion of the data from domain  $\lambda$  as  $\mathcal{P}_\lambda = \frac{\mathcal{N}_\lambda}{\mathcal{N}_{total}}$ , where  $\mathcal{N}_\lambda$  denote the number of data samples from domain  $\lambda$  and  $\mathcal{N}_{total}$  denote the total number of training samples. We then ensemble domain-specific models  $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k$  into a general domain-invariant model  $\mathcal{M}$  by computing  $\mathcal{M} = \mathcal{P}_1 \cdot \mathcal{M}_1 + \mathcal{P}_2 \cdot \mathcal{M}_2 + \dots + \mathcal{P}_k \cdot \mathcal{M}_k$ . Note that  $\mathcal{M}$  and each  $\mathcal{M}_\lambda$  are of size  $n \times \mathcal{D}$ .

### E. Hardware Optimizations

Edge-based ML applications often involve strict latency and performance requirements. We apply the following hardware-aware optimizations into the implementation of our proposed DOMINO to maximize its performance and robustness:

- **Multithreading.** DOMINO involves several implicit parallel processing opportunities. We utilizes multi-threads to parallelize operations including random basis generation, basis regeneration, encoding, and vector normalization.
- **Tiled matrix multiplication.** Several steps in DOMINO are naturally matrix operations, e.g., encoding and cosine similarity. We tile matrix operations with memory hierarchy in mind to optimize for the best local memory locality.

TABLE I  
DETAILED BREAKDOWNS OF DATASETS ( $\mathcal{N}$ : NUMBER OF DATA SAMPLES)

DSADS [22]		USC-HAD [23]		PAMAP2 [24]	
Domains	$\mathcal{N}$	Domains	$\mathcal{N}$	Domains	$\mathcal{N}$
Domain 1	2,280	Domain 1	8,945	Domain 1	5,636
Domain 2	2,280	Domain 2	8,754	Domain 2	5,591
Domain 3	2,280	Domain 3	8,534	Domain 3	5,806
Domain 4	2,280	Domain 4	8,867	Domain 4	5,660
		Domain 5	8,274		
Total	9,120	Total	43,374	Total	22,693

- **Kernel fusion.** We create custom kernels to fuse original kernels to reduce kernel invocation overheads.
- **Quantization.** DOMINO supports custom bitwidth to leverage low-bitwidth functional units on hardware platforms to further improve computational efficiency.

### III. EVALUATION

Taking human activity recognition as the application use-case, we evaluate DOMINO on widely-used multi-sensor time series datasets DSADS [22], USC-HAD [23], PAMAP2 [24]. Domains are defined by subject grouping chosen based on subject ID from low to high. The data size of each domain in each dataset is demonstrated in TABLE I. We compare DOMINO with (i) two SOTA CNN-based domain generalization (DG) algorithms: Representation Self-Challenging (RSC) [25] and AND-mask [26], and (ii) SOTA HDC without domain generalization capability [27] (BaselineHD). Our evaluations include leave-one-domain-out (LODO) performance, learning efficiency on both server CPU and resource-constrained devices, and robustness against hardware noises. The CNN-based DG algorithms are trained with TensorFlow, and we utilize the common practice of grid search to identify the best hyper-parameters for each model. The results of BaselineHD are reported in two dimensionality: (i) *Physical dimensionality* ( $\mathcal{D} = 0.5k$ ) of DOMINO, a compressed dimensionality designed for resource-constrained computing platforms, (ii) *Effective dimensionality* ( $\mathcal{D}^* = 4k$ ), defined as the sum of the physical dimensions ( $\mathcal{D}$ ) of DOMINO with all the regenerated dimensions throughout the retraining iterations. Mathematically,  $\mathcal{D}^* = \mathcal{D} + \mathcal{D} \times \mathcal{R} \times \text{Number of Iterations}$ , where  $\mathcal{R}$  is the regeneration rate. We also explore the hyperparameter design space of DOMINO to identify optimal hyperparameters.

#### A. Experimental Setup

To evaluate the performance of DOMINO on both high-performance computing environments and resource-limited devices, we include results from the following platforms:

- **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, TDP 120 W.
- **Embedded CPU:** Raspberry Pi 3 Model 3+ (quad-core ARM A53 @1.4GHz), 1 GB LPDDR2 memory, Debian 11, Python 3.9.2, PyTorch 1.13.1, TDP 5 W.
- **Embedded GPU:** Jetson Nano (quad-core ARM A57 @1.43 GHz, 128-core Maxwell GPU), 4 GB LPDDR4 memory, Python 3.8.10, PyTorch 1.13.0, CUDA 10.2, TDP 10 W.

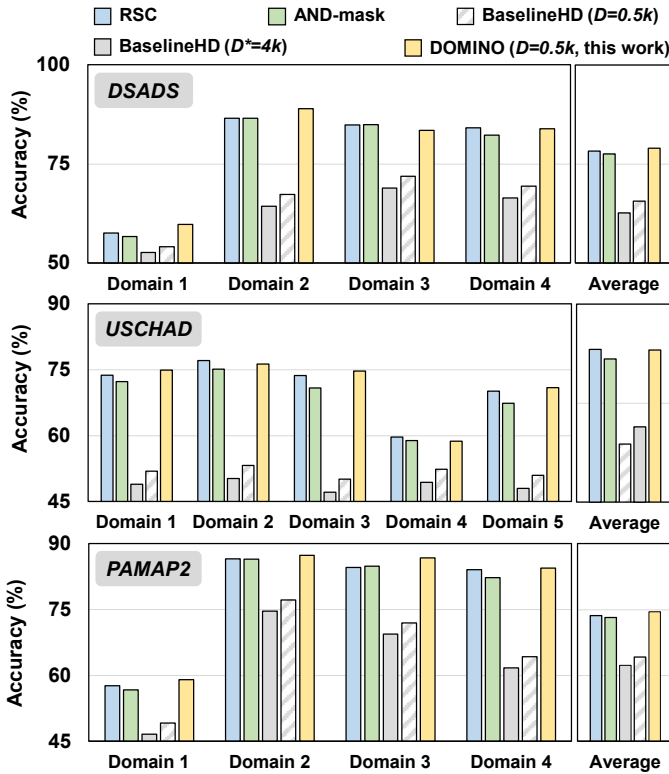


Fig. 6. LODO Accuracy of DOMINO and CNN-based Approaches

### B. Data Preprocessing

We describe the data processing steps for each dataset primarily focusing on data segmentation and domain labeling. For specific details, please refer to their respective papers.

**DSADS [22]:** The Daily and Sports Activities Dataset includes 19 activities performed by eight subjects. Each data segment is a non-overlapping five-second window sampled at 25Hz. Four domains are formed with two subjects each.

**USC-HAD [23]:** The USC human activity dataset includes 12 activities performed by 14 subjects. Each data segment is a 1.26-second window sampled at 100Hz with 50% overlap. Five domains are formed with three subjects each.

**PAMAP2 [24]:** The Physical Activity Monitoring dataset includes 18 activities performed by nine subjects. Each data segment is a 1.27-second window sampled at 100Hz with 50% overlap. Four domains, excluding subject nine, are formed with two subjects each. We also removed invalid and irrelevant data as suggested by the authors. Lastly, we only retain common activity labels (1, 2, 3, 4, 12, 13, 16, and 17).

### C. Accuracy

The accuracy of the LODO classification is shown in Fig. 6. The accuracy of Domain  $i$  means that the model is trained with data from all other domains and tested on data from Domain  $i$ . This accuracy score indicates the generalization capability of a trained model to data from unseen distributions. DOMINO outperforms SOTA CNN-based DG approaches by achieving on average 0.96% higher accuracy than RSC and 2.04% higher accuracy than AND-mask. DOMINO also exhibits 11.70% higher accuracy than BaselineHD ( $D^* = 4k$ ) and 15.93%

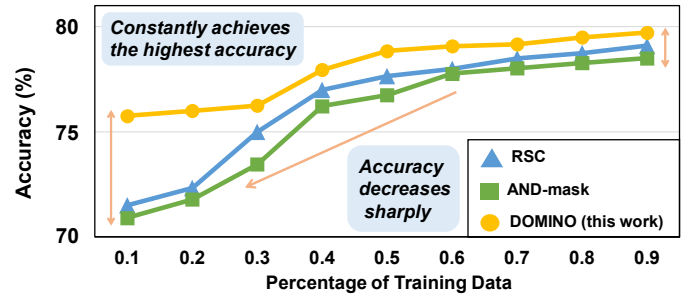


Fig. 7. Comparing Performance on Partial Training Data

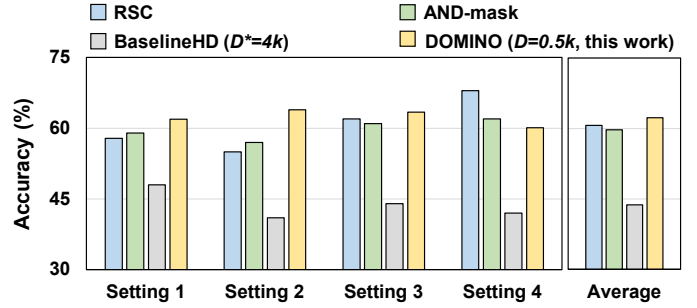
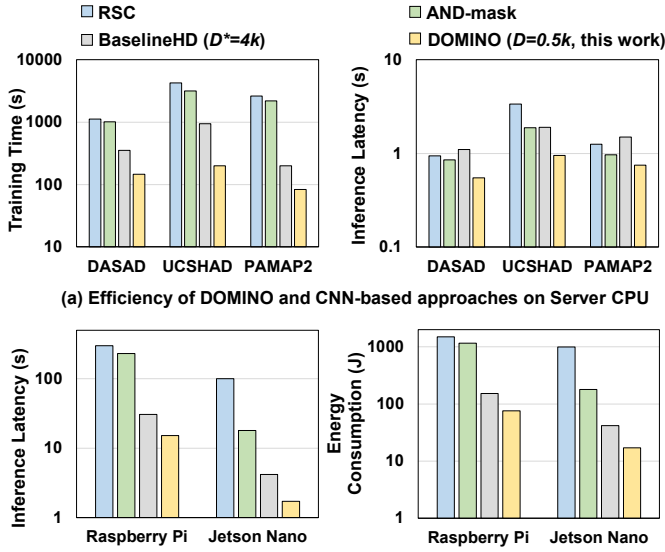


Fig. 8. Comparing Performance on Imbalanced Training Data

higher accuracy than BaselineHD ( $D = 0.5k$ ). This shows DOMINO effectively filters out domain-variant dimensions and thereby improves model generalization capabilities. Additionally, with the informative time series encoding technique and our dynamic regenerating method, DOMINO delivers reasonable accuracy with notably lower dimensionalities.

**Accuracy with Partial Training Data:** In practical implementation, edge-based ML applications frequently encounter scenarios wherein only a limited portion of data is labeled. We exhibit the performance of DOMINO with decreasing amounts of labeled data on the dataset DSADS. We randomly sample a portion (ranging from 10% to 90%) of training data from all the domains except Domain 5 for training, and evaluate our model by using data from Domain 5 for inference. As shown in Fig. 7, with the decreasing of data size, the learning accuracy of SOTA CNN-based DG algorithms decreases sharply while DOMINO is capable of constantly delivering significantly better performance. In particular, when we only use 10% of the training data, DOMINO demonstrates 5.81% higher accuracy than AND-mask and 4.90% high accuracy than RSC.

**Accuracy with Imbalanced Training Data:** Imbalanced training data, where certain domains contribute a disproportionately larger portion of the data while other domains are represented by considerably smaller amounts, often severely degrades model performance. We evaluate the performance of DOMINO when learning from highly imbalanced training data using the dataset USC-HAD as demonstrated in Fig. 8. In Setting  $\gamma$  ( $1 \leq \gamma \leq 4$ ), we construct the training data by randomly sampling 70% of the data from Domain  $\gamma$  and the remaining 30% from all other domains except Domain 5. We evaluate our trained model by using data from Domain 5 for inference. DOMINO outperforms SOTA CNN-based DG approaches by exhibiting on average 1.18% and 2.58% higher accuracy than RSC and AND-mask, respectively.



(b) Efficiency of DOMINO and CNN-based approaches on Embedded Platforms  
Fig. 9. (log scale) Comparing Efficiency on Server GPU and Edge Platforms

#### D. Efficiency

**Efficiency on Server CPU:** For fairness, we compare the learning efficiency of DOMINO with RSC, AND-mask, and BaselineHD ( $\mathcal{D} = 4k$ ) on the server CPU. For each dataset, each domain consists of roughly similar amounts of data as detailed in TABLE I; thus, we show the average runtime of training and inference for all the domains. As demonstrated in Fig. 9(a), DOMINO exhibits  $16.34\times$  faster training than RSC and  $14.17\times$  faster training than AND-mask. Additionally, DOMINO delivers  $2.89\times$  faster inference than RSC and  $1.97\times$  faster inference than AND-mask. Such notable higher learning efficiency is thanks to the highly parallel matrix operations on high-dimensional space and the notably faster convergence of HDC. Compared to BaselineHD ( $\mathcal{D}^* = 4k$ ), DOMINO delivers considerably higher accuracy without sacrificing much training efficiency as shown in Fig. 6 and Fig. 9. DOMINO also achieves  $1.93\times$  faster inference than BaselineHD ( $\mathcal{D}^* = 4k$ ) since it requires considerably lower physical dimensionality ( $\mathcal{D}$ ) and thereby reduces large amounts of unnecessary computations.

**Efficiency on Embedded CPU and GPU:** To further understand the performance of DOMINO on resource-constrained platforms, we evaluate the efficiency of DOMINO, RSC, and AND-mask 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. 9(b) shows the average inference latency for each algorithm processing each domain in the DSADS dataset. DOMINO outperforms CNN-based DG algorithms by providing speedups  $19.79\times$  compared to RSC and  $15.31\times$  compared to AND-mask on Raspberry Pi. DOMINO also delivers  $58.44\times$  faster inference than RSC and  $10.49\times$  faster inference than AND-mask on Jetson Nano. Additionally, DOMINO exhibits significantly less energy consumption, indicating that DOMINO can run efficiently on energy-constrained platforms.

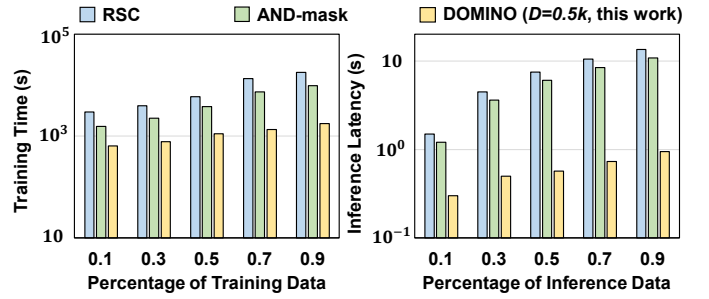


Fig. 10. Comparing Scalability Using Different Size of Data

Hardware Error		1.0%	2.0%	5.0%	10.0%	15.0%
DOMINO	RSC	2.1%	7.3%	11.2%	20.4%	29.7%
	AND-mask	2.9%	8.7%	13.8%	28.1%	35.2%
	1 bit	0.0%	0.0%	1.0%	3.1%	4.1%
	2 bits	0.0%	0.5%	1.6%	4.8%	8.0%
	4 bits	0.2%	1.0%	2.9%	7.4%	11.7%
	8 bits	1.4%	3.6%	5.1%	12.8%	17.6%

Fig. 11. Comparing Quality Loss Under Hardware Errors

#### E. Data Size Scalability

We exhibit the scalability of DOMINO and SOTA CNN-based DG approaches using various training data sizes (percentages of the full dataset). As shown in Fig. 10, with the increasing size of the training dataset, DOMINO is capable of maintaining high efficiency in both training and inference with a sub-linear growth in execution time. In contrast, the training and inference time of CNN-based algorithms increases notably faster than DOMINO. This indicates that DOMINO is capable of providing timely and scalable DG solutions for both high-performance and resource-constrained computing devices.

#### F. Robustness Against Hardware Noises

Leveraging encoded data points on high-dimensional space, DOMINO exhibits ultra-robustness against noise and failures, ensuring effective performance of DG classification tasks on noisy embedded devices. In particular, each hypervector stores information across all its components and no component is more responsible for storing any more information than another. We compare the robustness of DOMINO with SOTA CNN-based DG algorithms against hardware noise by showing the average quality loss under different percentages of hardware errors in Fig. 11. The error rate refers to the percentage of random bit flips on memory storing CNNs and DOMINO models. For fairness, all CNNs weights are quantized to their effective 8-bit representation. In CNNs, random bit flip results in significant quality loss as corruptions on most significant bits can cause major weight changes. In contrast, DOMINO provides significantly higher robustness against noise due to its redundant and holographic distribution of patterns with high dimensionality. Specifically, all dimensions equally contribute to storing information; thus, failures on partial data will not cause the loss of entire information. DOMINO exhibits the maximum robustness using hypervectors in 1-bit precision, that is on average 10.93% higher robustness than AND-mask and 8.62% higher robustness than RSC. Increasing precision lowers the robustness of DOMINO since random flips on more significant bits will cause more loss of accuracy.



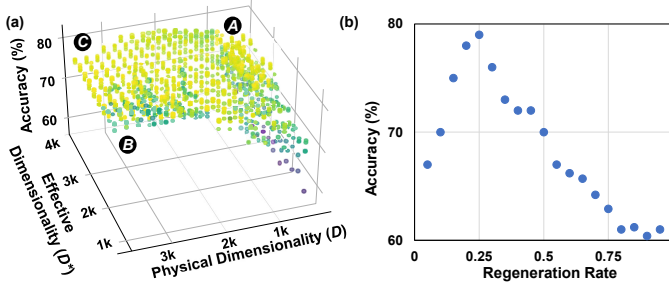


Fig. 12. Relation Between Accuracy and (a) Physical Dimensionality ( $D$ ) and Effective Dimensionality ( $D^*$ ) (b) Regeneration Rate ( $R$ )

### G. Design Space Exploration

To understand the behavior of DOMINO in different hyperparameter settings, we conduct a design space exploration traversing all possible combinations of different physical dimensionality ( $D$ ), effective dimensionality ( $D^*$ ), and regeneration rate ( $R$ ). We observe that a larger effective dimensionality ( $D^*$ ) play significant roles in enhancing the model performance. For instance, in Fig. 12(a), though starting from a much smaller  $D$ , the DG accuracy of points near **A** is notably higher than points near **B** and is comparable to points near **C**, as the disadvantage of utilizing a very limited physical dimensionality can be compensated by more training iterations and larger effective dimensionality. This characteristics is extremely appealing since using models with compressed dimensionalities significantly reduces unnecessary computations involved in training and inference and can be more resource-efficient for edge-based ML applications. Mathematically,  $\text{Number of Iterations} = \frac{D^* - D}{D \times R}$ . This indicates when  $R$  and  $D$  are fixed, models with larger  $D^*$  go through more iterations of filtration and regeneration of domain-variant dimensions and potentially perform better. We also learn the relation between accuracy and  $R$  as shown in Fig. 12(b) by fixing  $D$  and  $D^*$  to 0.5k and 4k, respectively. We observe DOMINO achieves optimal performance when  $R$  is around 0.25. A too small  $R$  will cause failure in filtering out the majority of domain-invariant dimensions, while a too large  $R$  will lead to performance degradation due to excessively eliminating and regenerating informative dimensions.

## IV. RELATED WORKS

### A. Distribution Shift

Distribution shift (DS), where a model is deployed on a data distribution different from what it was trained on, poses significant robustness challenges in real-world ML applications [5], [9], [28]. Various innovative ideas have been proposed to mitigate this issue, and can be majorly categorized as *domain generalizations* (DG) and *domain adaptations* (DA). DA generally utilizes unlabeled or sparsely labeled data in target domains to quickly adapt a model trained in different source domains [29], [30]. In contrast, most DG approaches aim to build models by extracting domain-invariant features across known domains [30]–[32]. Existing DG are primarily based on CNNs [25], [26], [30], [33], [34] and

target image classification and object detection tasks. However, these approaches often rely on multiple convolutional and fully-connected layers, requiring intensive computations and iterative refinement. This can often be challenging to implement on resource-constrained platforms considering the memory and power limitations [35]. Our proposed DOMINO is the first HDC-based DG algorithm aiming to provide a more resource-efficient and hardware-friendly DG solution for today's edge-based ML applications. It utilizes encoded data points on high-dimensional space to dynamically identify and regenerate domain-variant dimensions, thereby enhancing model generalization capability.

### B. Hyperdimensional Computing

Prior studies have exhibited enormous success in various applications of HDCs, such as brain-like reasoning [36], bio-signal processing [37], and cyber-security [20]. A few endeavors have been made toward utilizing HDC for time series classification, including data from Electroencephalography (EEG) and Electromyography (EMG) sensors [16], [38], [39]. However, existing HDCs do not consider the challenge of distribution shift. For instance, TempHD [16] relies on historical EEG data from an individual to make inferences for the same individual. This can be a detrimental drawback, especially in the deployment of embedded AI applications. Hyperdimensional Feature Fusion [8], a recently proposed algorithm for out-of-distribution (OOD) sample detection, maps information from multiple layers of DNNs into class-specific vectors in high-dimensional space to deliver ultra-efficient performance. However, its proposed algorithm remains to rely on resource-intensive multi-layer DNNs, and a systematic way to deal with OOD samples for DG has not yet been proposed. In contrast, we propose DOMINO, an HDC-based DG learning framework that fully leverages the highly efficient and parallel matrix operations on high-dimensional space to filter out dimensions representing domain-variant information.

## V. CONCLUSION

In this paper, we propose DOMINO, an innovative HDC-based DG algorithm for noisy multi-sensor time series classification. Our evaluations demonstrate that DOMINO achieves notably higher accuracy than CNN-based DG approaches, especially when learning from partially labeled data and highly imbalanced data. Additionally, DOMINO provides a hardware-friendly solution for both high-performance computing devices and resource-constrained platforms by delivering significantly faster training and inference on both server CPU and embedded platforms. Leveraging holographic pattern distributions on high-dimensional space, DOMINO also exhibits considerably higher robustness against hardware errors than CNN-based approaches, bringing unique advantages in performing learning tasks on noisy and unstable edge devices.

## VI. ACKNOWLEDGEMENT

This work was partially supported by the National Science Foundation (NSF) under award CCF-2140154.



## REFERENCES

- [1] J. Wang, S. Huang, and M. Imani, "Disthd: A learner-aware dynamic encoding method for hyperdimensional classification," *arXiv preprint arXiv:2304.05503*, 2023.
- [2] R. R. Shrivastwa *et al.*, "A brain-computer interface framework based on compressive sensing and deep learning," *IEEE Consumer Electronics Magazine*, 2020.
- [3] N. Rashid *et al.*, "Template matching based early exit cnn for energy-efficient myocardial infarction detection on low-power wearable devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2022.
- [4] B. U. Demirel *et al.*, "Neural contextual bandits based dynamic sensor selection for low-power body-area networks," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022.
- [5] K. Dong and T. Ma, "First steps toward understanding the extrapolation of nonlinear models to unseen domains," in *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022. [Online]. Available: <https://openreview.net/forum?id=lfS4KqfrY1>
- [6] E. H. Pooch *et al.*, "Can we trust deep learning based diagnosis? the impact of domain shift in chest radiograph classification," in *Thoracic Image Analysis: Second International Workshop, TIA 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 8, 2020, Proceedings 2*. Springer, 2020.
- [7] A. Subbaswamy *et al.*, "From development to deployment: dataset shift, causality, and shift-stable models in health ai," *Biostatistics*, 2020.
- [8] S. Wilson *et al.*, "Hyperdimensional feature fusion for out-of-distribution detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023.
- [9] I. Gulrajani *et al.*, "In search of lost domain generalization," in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://openreview.net/forum?id=IQdXeXDoWtI>
- [10] M. Wang *et al.*, "Deep visual domain adaptation: A survey," *Neurocomputing*, 2018.
- [11] S. Zhao *et al.*, "A review of single-source deep unsupervised visual domain adaptation," *Transactions on Neural Networks and Learning Systems*, 2020.
- [12] G. Wilson *et al.*, "Multi-source deep domain adaptation with weak supervision for time-series sensor data," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020.
- [13] Y. Qin *et al.*, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.
- [14] Y. Su *et al.*, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019.
- [15] S. Hochreiter *et al.*, "Long short-term memory," *Neural computation*, 1997.
- [16] Y. Ni *et al.*, "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022.
- [17] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*. IEEE, 2019.
- [18] L. Ge *et al.*, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, 2020.
- [19] Z. Zou *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *SC*, 2021.
- [20] J. Wang, H. Chen, M. Issa, S. Huang, and M. Imani, "Late breaking results: Scalable and efficient hyperdimensional computing for network intrusion detection," *arXiv preprint arXiv:2304.06728*, 2023.
- [21] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in neural information processing systems*, 2007.
- [22] B. Barshan *et al.*, "Recognizing daily and sports activities in two open source machine learning environments using body-worn sensor units," *The Computer Journal*, 2014.
- [23] M. Zhang *et al.*, "Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors," in *Proceedings of the 2012 ACM conference on ubiquitous computing*, 2012.
- [24] A. Reiss *et al.*, "Introducing a new benchmarked dataset for activity monitoring," in *16th international symposium on wearable computers*. IEEE, 2012.
- [25] Z. Huang *et al.*, "Self-challenging improves cross-domain generalization," in *Computer Vision-ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020.
- [26] G. Parascandolo *et al.*, "Learning explanations that are hard to vary," *arXiv preprint arXiv:2009.00329*, 2020.
- [27] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021.
- [28] R. Palakkadavath *et al.*, "Improving domain generalization with interpolation robustness," in *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022.
- [29] G. Csarka *et al.*, *Domain adaptation in computer vision applications*. Springer, 2017.
- [30] D. Li *et al.*, "Learning to generalize: Meta-learning for domain generalization," in *Proceedings of the AAAI conference on artificial intelligence*, 2018.
- [31] H. Li *et al.*, "Domain generalization with adversarial feature learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.
- [32] Q. Dou *et al.*, "Domain generalization via model-agnostic learning of semantic features," *Advances in Neural Information Processing Systems*, 2019.
- [33] Y. Ganin *et al.*, "Domain-adversarial training of neural networks," *The journal of machine learning research*, 2016.
- [34] S. Sagawa *et al.*, "Distributionally robust neural networks," in *International Conference on Learning Representations*, 2019.
- [35] J. Pan *et al.*, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, 2017.
- [36] P. Poduval *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, 2022.
- [37] A. Burrello *et al.*, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019.
- [38] A. Moin *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, 2021.
- [39] A. Rahimi *et al.*, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *International Conference on Rebooting Computing*. IEEE, 2016.