IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Multi-label classification with Hyperdimensional Representations

Rishikanth Chandrasekaran[1], Fatemeh Asgareinjad[1], Justin Morris[2], Tajana Rosing [1]
[1]University of California San Diego (e-mail: r3chandr@ucsd.edu, fasgarinejad@ucsd.edu, tajana@ucsd.edu)
[2]California State University San Marcos (e-mail: jmorris@csusm.edu)

ABSTRACT Hyperdimensional computing (HDC) is a computational paradigm that leverages the mathematical properties of high-dimensional vector spaces to manipulate data as symbolic entities using a set of neurally plausible operations. Although HDC has demonstrated remarkable success in cognitive tasks, its potential in complex applications such as multi-label classificati has yet to be explored. In this research paper, we introduce three approaches to multi-label classification that strike a balance between computational efficiency and accuracy, based on the complexity of the problem. The first approach we propose is Power Set HD, a transformation method that is ideal for small-scale multi-label classification with label cardinality less than four and label set size less than ten. The second approach, One-vs-All HD, is another transformation method that is suitable for slightly more complex tasks with higher label cardinality, providing a better efficiency-accuracy trade-off over Power Set HD. However, due to the expensive linear complexity scaling of One-vs-All HD, we propose a novel neural approach called TinyXML HD for extreme scale tasks. This method learns hyperdimensional representations by decomposing the learning problem into multiple sub-problems, which are solved neurally through gradient-based optimization. Importantly, TinyXML HD fixes the output size of the model to the dimensionality of the hypervector, regardless of the label size, thereby scaling only by a small constant when evaluated on datasets with extremely large label spaces. Our approaches offer a valuable trade-off between computational efficiency and accuracy. We show that our methods provide a speedup of 16-60x on state of the art datasets, while maintaining comparable accuracy. Furthermore, our methods yield models that are 56x smaller on medium-scale tasks and up to 836x smaller on extreme-scale datasets, which is a significant reduction in model size while still achieving high accuracy.

INDEX TERMS Hyperdimensional Computing, Multi-label classification, Deep Learning

I. INTRODUCTION

Hyperdimensional computing (HDC) is an emerging paradigm of computing that offers a promising alternative to traditional machine learning approaches. In recent years, HDC has garnered significant interest due to its low computational overhead and hardware-friendly nature [1], [2]. HDC employs low-precision sparse representations and simple arithmetic operations to manipulate high-dimensional vectors, making it amenable to hardware acceleration. The independent and identically distributed (iid) nature of these representations further enables efficient parallelization, leading to improved computational efficiency. There is a large body of works showing benefits of HDC acceleration in hardware for various applications in IoT [3]–[5] and Machine Learning [6]–[8]. As a result, HDC has gained popularity in various domains, including natural language processing [9], [10], biomedical applications like DNA pattern matching [11] and protein alignment [12] and robotics [13]. Despite its success, the applicability of HDC for complex tasks like multi-label classification, which has real-world applications in recommender systems and document classification, has not been explored.

Our research presents the first comprehensive exploration of multi-label learning problems utilizing HDC representations. We introduce three novel approaches that strike an optimal balance between computational efficiency and accuracy. Our first approach, Power Set HD, is a transformation method that achieves exceptional accuracy and efficiency on datasets with small label spaces and a limited subset of possible label combinations. For small datasets with larger label cardinality ($\geq$ 4), we propose One-vs-All HD, which reduces the exponential complexity scaling of Power Set HD to a linear scale on the label set size, making it ideal for datasets with label sizes up to 30. In addition, we present TinyXML HD, a

neural approach to learning mappings between hypervectors by decomposing the problem into multiple sub-problems. TinyXML HD fixes the output dimensionality of the model independent of the label size or cardinality, making it an ideal candidate for extreme multi-label classification problems.

HDC leverages neurally plausible representations of data and associates abstract concepts with high-dimensional vectors to perform complex cognitive tasks. The two fundamental operations of HDC are "bundling" and "binding" [2]. Bundling (denoted by $\oplus$) is used to represent multiple symbolic entities (hypervectors) using a single hypervector, while binding (denoted by $\otimes$) associates one entity with another.

We leverage the Multiply Add Permute (MAP) architecture proposed by Gayler [14], which uses bipolar representations for HDC. MAP represents data using high-dimensional vectors $X \in \{+1, -1\}^D$ called hypervectors. Gayler demonstrated that by assigning hypervectors with a conceptual meanin, we can represent conceptual relationships using these operators. For example, the sentence *"Yoda is a Jedi and Leia is a princess"* can be represented as $H = \text{Yoda} \otimes \text{Jedi} \oplus \text{Leia} \otimes \text{princess}$. HDC also allows us to query and reason about expressions. For instance, to find who is a Jedi, using the inverse operator $\tilde{\text{Jedi}}$ we can simply compose $H \otimes \tilde{\text{Jedi}} \approx \text{Yoda}$, which results in a hypervector that is approximately equal to Yoda.

Gradient-based neural methods have demonstrated tremendous success in various learning tasks [15]. They provide a systematic approach to finding the minima of a function [16] and can be efficiently computed provided the function is differentiable. The MAP framework uses element-wise products or additions that are themselves differentiable, however, the quantization step that follows is not differentiable. There are other HDC models with fully differentiable operations such as Holographic Reduced Representations (HRR) [17] which have been studied by using a neural gradient-based approach in the context of multi-label classification like [18]. However, the HRR framework requires Fast Fourier Transform (FFT) operations which increase complexity. The MAP model in contrast, uses simple operations that can be accelerated using efficient bit-wise operations.

In [18], the authors utilized symbolic hypervectors to represent the labels and employed neural methods to learn a mapping from the instance space to the label space. To simplify the learning problem, we instead embedded both inputs and labels in the same high-dimensional vector space, which can be learned more easily than mapping across different vector spaces [19]–[21]. This is because the model does not need to learn complex transformations to map between spaces, reducing the complexity of the learning problem. After embedding inputs and labels as hypervectors in the same vector space, TinyXML HD learns a mapping between the two using a 1-D convolutional neural network designed for processing hypervectors.

In this work we introduce three methods that show the potential of HDC to solve multi-label classification problems across the entire spectrum of complexity - small, medium, and large, as described below:

- The first approach, Powerset HD, is suitable for small-scale multi-label classification, where each possible label combination is instantiated as a separate binary learning problem, resulting in exponential scaling over label size. This approach yields high-accuracy models that scale well for datasets with a few label combinations.
- The second approach, One-vs-All HD, is another transformation method that relaxes the exponential scaling of Powerset HD to linear scaling over label size, resulting in models that are efficient and accurate for datasets with a label set size of up to 30. Beyond this limit, the training time increases significantly, making this method less suitable.
- For extreme-scale multi-label problems, we propose TinyXML HD, which utilizes a 1-D convolutional neural network to learn hypervector representations. By having a fixed output dimensionality independent of the label complexity, TinyXML HD achieves remarkable speedups in training. However, due to the relatively expensive convolution operations, the first two approaches provide a better trade-off between computational efficiency and accuracy for smaller size datasets.
- Through rigorous evaluations on real-world datasets, we demonstrate the superiority of our proposed methods. Powerset HD and One-vs-All HD offer up to 60x speedup on small-scale datasets, while TinyXML HD is 56x smaller compared to the state-of-the-art on medium-scale datasets and up to 836x smaller on extreme-scale datasets, all while maintaining comparable accuracy.

The rest of the article is organized as follows. We first review related work in II, highlighting the differences between our approach and existing methods. In section III, we provide an overview of HDC helpful for understanding the rest of the article. We split the problem into two variants: micro multi-label classification and extreme multi-label classification. We first tackle the micro multi-label classification in IV, where we discuss two simple problem transformation techniques and examine their performance on trivial learning problems in VI-A. We provide an overview of the Extreme Multi-Label Classification in V, followed by Section V-A where we detail a new encoding method for representing text data as hypervectors. We then present our novel HDC convolution operator and neuro-symbolic approach in V-B, detailing its formulation and demonstrating its effectiveness in Section VI-B.

II. RELATED WORK

Hyperdimensional computing (HDC) is an emerging field that aims to address the limitations of traditional computing paradigms by leveraging high-dimensional vector representations to perform complex cognitive and machine learning tasks. This section presents a brief summary of various HDC

works, highlighting their contributions to both cognitive tasks and machine learning tasks. We also present a brief survey of gradient based algorithms and multi-label classification for the readers benefit.

A. HDC FOR COGNITIVE & LEARNING TASKS

Kanerva introduced the foundational concept of a "hyperdimensional computer", which efficiently stores and retrieves information using large, sparse binary vectors [22]. This model exhibited robustness and efficiency in cognitive tasks, inspiring further research in HDC. Gallant et al. explored HDC in natural language understanding and reasoning, successfully capturing semantic relations in text and showcasing its potential for large-scale knowledge representation [23]. Rachkovskij expanded HDC's application to image processing and recognition, demonstrating pattern recognition capabilities with high accuracy and noise robustness [24]. In [25] Anthony et al. develops a theoretical framework for HDC and details the mathematical properties of HDC encoding methods.

In recent years, hyperdimensional computing (HDC) has emerged as a promising paradigm for machine learning, such as classification, regression, and reinforcement learning. Lai et al. employed HDC for classification tasks, developing a high-dimensional classifier that achieved competitive performance with reduced computational complexity [26]. Imani et al. applied HDC to regression problems, proposing a high-dimensional computing framework that provided accurate and efficient regression models with minimized computational overhead [27]. Goudarzi et al. explored HDC in reinforcement learning, developing a state representation and policy learning approach that demonstrated effectiveness in various environments [28]. Imani et al. proposed HDCluster, an accurate clustering algorithm for high-dimensional datasets using hyperdimensional computing [29]. In GENERIC [3], Khaleghi et al. proposed a novel and efficient method for learning on edge devices using hyperdimensional computing for a wide range of applications. The method utilizes hardware-friendly hyperdimensional vector representations and an optimized training algorithm to reduce computation and storage requirements while maintaining high accuracy. Guo et al. [7] proposed using hypervectors to represent users and items and performs a set of associative and distributive operations on these vectors to compute recommendations. The paper presents three different methods for generating recommendations, including one that combines hyperdimensional computing with matrix factorization. Asgarinejad et al. [30] developed a method for epilepsy detection using EEG signals. They demonstrate using real-world data that HDC approaches outperforms state-of-the-art methods like Support Vector Machines (SVM) [31] and Convolutional Neural Networks (CNN) [32].

B. GRADIENT BASED HDC METHODS

Gradient-based methods in hyperdimensional computing (HDC) have garnered interest due to their potential for addressing optimization challenges in high-dimensional spaces. These methods extend the capabilities of HDC by incorporating gradient information to guide the learning process. For instance, Frady and Sommer introduced a gradient-based HDC framework, which allowed the use of optimization algorithms such as gradient descent and backpropagation in HDC settings [33]. In a subsequent work, Frady et al. proposed a method for gradient-based learning in HDC that utilized iterative projections and local linearizations to facilitate learning in high-dimensional spaces [34]. Building upon these developments, Wang et al. presented a gradient-based HDC algorithm for clustering and classification tasks, which employed a convex optimization formulation to enhance HDC's performance in these applications [35]. Moreover, Su et al. developed a gradient-based HDC algorithm for deep learning, illustrating the potential of gradient-based methods in improving the robustness and expressiveness of HDC models [36]. Recently, Zhou et al. presented a gradient-based HDC framework for unsupervised learning, focusing on clustering and dimensionality reduction tasks [37]. These studies highlight the increasing importance of gradient-based methods in HDC and their potential in addressing various learning tasks in high-dimensional spaces.

While these methods have shown promise in addressing optimization challenges in high-dimensional space, they introduce additional complexity in order to facilitate backpropogation through the HDC operations. For example, Frady and Somer's work involves iterative projections and local linearizations which can be expensive. Similarly Wang's convex optimization formulation for clustering and classification tasks can result in increased computational overhead [35]

Prior works have also explored the use Holographic Reduced Representations (HRR), a family models for gradient based learning tasks. A notable attempt to capitalize on the symbolic properties of HRR was made by Nickel et al. [38], who utilized binding operations to link elements within a knowledge graph. Their approach served as an embedding mechanism that merged two vectors of information without increasing the dimensionality of the representation, as opposed to concatenation which doubles the dimension. In a more recent study, Liao and Yuan [39] employed circular convolution as a substitute for standard convolution to decrease model size and inference time, albeit without leveraging the symbolic properties inherent to HRRs. Although Danihelka et al. [40] claimed to incorporate HRR into an LSTM, their methodology simply augmented an LSTM with complex weights and activations, and did not genuinely implement HRR due to the absence of circular convolution.

C. MULTI-LABEL CLASSIFICATION

The seminal works of multi-label classification emerged in the early 2000s with the introduction of the problem and initial approaches [41]. Since then a wide range of algorithms and techniques have been proposed to tackle this problem such as problem transformation methods [42], and ensemble methods [43], [44].

Among the various methods for multi-label classification, problem transformation methods have gained considerable attention. These techniques transform the multi-label problem into one or more single-label problems, which can then be addressed using traditional machine learning classifiers. One popular approach is the Binary Relevance (BR) method [45], which independently trains a binary classifier for each label. Another problem transformation method is the Label Powerset (LP) method [45], which treats each unique combination of labels as a single class in a multi-class problem. To address the shortcomings of BR and LP, researchers have proposed various ensemble and hybrid techniques. These include the Random k-Labelsets (RAkEL) method [46], which constructs multiple LP classifiers on random label subsets, and the Classifier Chains (CC) method [47], which constructs a chain of binary classifiers while preserving label correlations. Apart from problem transformation methods, other multi-label classification techniques include algorithm adaptation methods, which modify single-label algorithms to handle multi-label data directly. Examples of such methods are the Multi-Label k-Nearest Neighbors (ML-kNN) algorithm [48], and the Multi-Label Decision Trees (MLDT) [49].

Extreme multi-label classification (XML) is a specialized form of multi-label classification, characterized by a large number of labels and instances. XML has attracted significant research attention due to its relevance in numerous real-world applications, such as large-scale document classification [50], image annotation [51], and gene function prediction [52]. Early approaches for XML include the FastXML algorithm [53], PfastreXML algorithm [54], and the Parabel algorithm [55]. Embedding-based methods, such as the SLEEC algorithm [50] and the AnnexML algorithm [56], have also been proposed for XML.

Deep learning approaches have shown considerable promise in XML tasks. Convolutional Neural Networks (CNNs) [57], Recurrent Neural Networks (RNNs) [58], and Transformer models [59] have been adapted for XML problems, demonstrating improved performance compared to traditional methods. Specifically, BERT [59] and its variants have been successfully applied to large-scale text classification tasks.

D. MOTIVATION AND OUR CONTRIBUTIONS

In the existing literature on hyperdimensional computing (HDC), the majority of studies have focused on small-scale learning problems. Ganesan et al. [18] examined the extreme multi-label text classification task, but other works have yet to explore the scalability of HDC techniques in addressing large-scale machine learning problems in real-world applications. Our research aims to bridge this gap by investigating the application of HDC to a demanding, industrial-scale learning problem.

State-of-the-art deep learning models for multi-label classification, such as X-Transformer [60] and LightXML [61], comprising millions of parameters, necessitate days of training to achieve optimal performance. Our objective in this work is to examine HDC's potential for reducing this training time, thereby offering a more balanced trade-off between computational efficiency and accuracy.

Ganesan et al. [18] proposed a method for extreme multi-label text classification that replaces the final classification layer of AttentionXML [62] and XML-CNN [63] with a fully connected layer that outputs a hypervector encoding the relevant label information for an instance. While they demonstrated that their proposed method achieves accuracy similar to the baseline implementations of AttentionXML and XML-CNN, there are two key areas to improve upon. First, their method uses the HRR binding operation, which is a circular convolution requiring Fast Fourier Transform [64], an expensive operation. Second, their method learns a mapping from the instance space (represented as one-hot encoding) to the label space (represented as HRR hypervectors), which is a harder learning problem requiring learning the projection across the vector spaces.

In contrast, our approach is based on the Multiplicative Addition Perturbation (MAP) model introduced by Gayler [9], which uses bi-polar representations with simple element-wise arithmetic operations that can be easily accelerated and parallelized on hardware. Additionally, we embed the inputs and labels both in the same high-dimensional vector space, thereby avoiding the need to learn complex transformations across vector spaces. Our proposed neural approach for learning high-dimensional representations not only avoids increased computational complexity but also reduces the compute cost by a factor of 200.

III. HYPERDIMENSIONAL COMPUTING BACKGROUND

Hyperdimensional computing (HDC) is an emerging paradigm of computing that describes a family of representations and operations using high-dimensional vectors called hypervectors [14], [17], [22]. The basic idea behind HDC is to represent structured or symbolic data using hypervectors and then provide a set of mathematical operations to manipulate these vectors like symbolic objects. These operations are associative, commutative, and distributive [65], they operate element-wise, allowing them to be performed in parallel, making HDC an attractive approach for implementing hardware-accelerated, energy-efficient computing.

Hypervectors are typically represented as binary or bipolar vectors in a high-dimensional space. Mathematically, a hypervector is represented by a vector $X \in \{+1, -1\}^D$ where D is the dimensionality of the vector space. The dimensionality of the hypervector is often much larger than the number of dimensions required to represent the data, enabling the vector to encode many concepts or attributes in a single representation. For instance, a hypervector representing an object might contain attributes such as color, shape, texture, and position.

A. HDC OPERATIONS

HDC provides three fundamental operations: bundling, binding and similarity check. These operations are implemented

differently in various HDC models, and we will briefly explain their usage under the Multiply-Add-Permute (MAP) model. In the MAP framework, hypervectors are bipolar and can be represented as $X = \{+1, -1\}^D$.

1) Bundling

The bundling operation is used to represent multiple symbolic entities using a single hypervector. This operation is denoted by the $\oplus$ symbol and can be expressed as:

$$\text{bundle}(X_1, X_2, ..., X_n) = X_1 \oplus X_2 \oplus ... \oplus X_n$$

where $X_1, X_2, ..., X_n$ are hypervectors representing the symbolic entities. The result of the bundling operation is a new hypervector that represents the combination of all the input entities. For MAP the bundling operation is a simple element-wise sum of the hypervectors. Under the similarity check metric defined below, the resultant hypervector is similar to its constituent hypervectors.

2) Binding

The binding operation is used to associate one entity with another and is denoted by the $\otimes$ symbol. The binding operation is defined as the element-wise multiplication of two hypervectors and can be expressed as:

$$\text{bind}(X, Y) = X \odot Y$$

where X and Y are the hypervectors representing the two entities to be associated. The result of the binding operation is a new hypervector that encodes the relationship between the two input entities. By the similarity metric, the resultant hypervector is orthogonal to the input hypervectors.

3) Similarity check

Finally, the similarity check operation is used to determine the degree of similarity between two hypervectors. The similarity check operation is defined as the dot product between two hypervectors and can be expressed as:

$$\text{similarity}(X, Y) = X \cdot Y$$

where X and Y are the two hypervectors to be compared. The result of the similarity check operation is a scalar value that represents the degree of similarity between the two hypervectors, with higher values indicating greater similarity.

Together, these operations provide a powerful and flexible approach to representing and manipulating symbolic data in a distributed and parallel fashion, enabling the development of novel machine learning algorithms and cognitive models.

B. HDC LEARNING

The HDC learning process involves the encoding of data and its inherent relationships within hypervectors. These vectors are then subjected to a set of mathematical operations, enabling the extraction of useful patterns and relationships within the data. Learning in HDC involves three steps: encoding the data, learning on the encoded data and inference.

Encoding: The first step in learning with HDC involves encoding the input data into high-dimensional hypervectors. The goal of this step is to create a distributed representation of the input data that can capture its semantic properties. Previous literature have proposed various techniques for encoding data into hypervectors each with different mathematical properties. One key distinction among these techniques is the way in which distance metrics are preserved within the encoded data when mapping to hypervectors. Here we provide a brief overview of our chosen encoding scheme, namely Random Projection Encoding (RPE) [25], which utilizes Gaussian distribution to generate the encoding vectors.

Given an input vector $X \in R^d$, we generate a set of random projection vectors $R_1, R_2, ..., R_K \in R^D$, where $D >> d$. The projection of X onto the k^{th} random projection vector is given by the dot product $X \cdot R_k$. The resulting set of K projections can be represented as a hypervector $H \in \{+1, -1\}^K$, where $H_i = sign(X \cdot R_i)$. Random projection encoding in HDC has been shown to preserve Euclidean distance in the original vector space, mapping it to angular distance in the high-dimensional space [25]. This similarity-preserving nature makes it suitable for encoding data by retaining complex relationships between them.

Training: In HDC, training typically involves two steps: one-shot training followed by iterative retraining. One-shot training involves representing each class with a centroid hypervector that is the average of hypervectors representing the training examples for that class. Retraining involves updating the centroid hypervectors using a simple perceptron-style algorithm [66] in an iterative process that runs until convergence. During retraining, the centroid hypervectors are updated when a sample is mispredicted, with updates applied to both the correct label and the mispredicted label.

Inference: The centroid vectors can be used to classify new data by measuring the similarity of the new data to each centroid vector. The class with the highest similarity measure is chosen as the predicted class for the new data point. Hamming distance similarity is used for binary hypervectors, while cosine similarity can be used for any other type of data.

IV. MULTI-LABEL OVA & POWERSET HD

Multi-label classification is a machine learning problem where an instance can belong to multiple classes simultaneously. Mathematically, it can be defined as follows: Let $\mathbf{x}$ be a feature vector representing an instance and $\mathbf{y}$ be a binary vector indicating the presence or absence of L possible class labels, where $y_i = 1$ indicates the instance belongs to the i^{th} class and $y_i = 0$ indicates otherwise. The goal of multi-label classification is to learn a mapping function $f(\cdot)$ that takes as input an instance $\mathbf{x}$ and outputs a binary vector of length L indicating the classes the instance belongs to.

The complexity of this task can be influenced by various factors, such as the number of labels, the label dependencies, and the label cardinality. To distinctly refer to the class of

problems with relatively small label spaces, we define a small scale variant of multi-label learning. This variant deals with datasets where the label space is simple and the number of instances is relatively small, resulting in label set sizes of less than 100. In Section V we discuss the characteristics of the most challenging variant of multi-label classification that focuses on very large problem sizes. Our prior work [67] laid the foundations of hyperdimensional multi-label classification by combining HDC with two well studied problem transformation techniques, One-vs-All [68] and Label Powerset [69], to solve micro size problems.

Problem transformation methods [45], [70], [71] have been proposed where the original multi-label problem is transformed into multiple single-label problems. Each transformed problem corresponds to one of the L class labels and involves training a binary classifier to distinguish instances that belong to that class from those that do not. The output of each binary classifier is then combined to obtain the final multi-label prediction. These methods can be further classified into three categories: 1) One-vs-All [68], 2) Label Powerset [69], and 3) Classifier Chains [70], each with their own advantages and disadvantages. We consider the first two methods due to their simpler nature which is appropriate for the micro size problems.

PowerSet & OvA HD involve learning multiple binary classifiers, and hence, share a common implementation strategy. The difference between the two approaches lies in the way the class hypervectors are set up. We begin by encoding each instance in the dataset into a symbolic hypervector using Random Projection Encoding [25] as explained in Section III. We then perform one-shot learning, which involves learning the centroid hypervectors, followed by iterative fine-tuning, as detailed in Section III. The specific differences between the powerset and OvA approaches in this implementation are explained below.

A. POWERSET HD

The label powerset transformation method defines each unique combination of labels as a distinct class, represented by a binary class vector. This makes it possible to use standard single-label classification algorithms to train models on multi-label data. Formally, given an instance $\mathbf{x}$ with L possible class labels, this method creates a new binary class vector $\mathbf{y}$ of length 2^L, representing all possible label combinations. For each unique combination of labels C_j, a binary label is assigned based on whether the combination is a subset of the original class labels of the instance $\mathbf{x}$, as shown in Equation 1:

$$\mathbf{y}_i = \begin{cases} 1 & \text{if } S_j \subseteq C_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $C_j \subseteq \mathbf{y}$ indicates that the class combination C_j is a subset of the original class labels of the instance $\mathbf{x}$. For example, if an instance has three possible class labels A, B, and C, then there are $2^3 = 8$ possible combinations of labels: $\{\emptyset, A, B, C, AB, AC, BC, ABC\}$.

While the label powerset method is simple and easy to understand, it suffers from the issue of class imbalance and scaling with the number of class labels, making it less practical for problems with large numbers of class labels. Nevertheless, it is still widely used as a baseline method for evaluating the performance of other more advanced multi-label learning methods.

To implement power set transformation with HDC, We create a centroid hypervector for every label combination resulting in 2^L centroid hypervectors. Retraining is done on each centroid hypervector individually as an independent binary classifier. During inference, we encode the test instance and compare it with each of the centroid hypervectors using a similarity check function. The closest centroid indicates the relevant label combination.

Compute Realization Cost of PowerSetHD: To estimate the storage size of the HDC model, we need to calculate the total number of hypervectors required to represent all possible label combinations, and then multiply that by the size of each hypervector in bits. The number of possible label combinations for a dataset with L labels is 2^{2L}, since each label can either be present or absent in a given combination. Let's consider the case of Delicious dataset where number of labels is $L = 983$, then the number of possible label combinations is $2^9 83$. To represent each hypervector as a 16-bit integer, we need 16 bits or 2 bytes per element. Since each hypervector has 1024 elements, the size of each hypervector in bytes is $2 \times 1024 = 2048$ bytes. Multiplying the number of hypervectors by the size of each hypervector gives us the total storage size required for the HDC model: $2^{983} \times 2048$ bytes/hypervector, which is equivalent to 1.4×10^{269}Terabytes. This is an enormous amount of storage, far beyond what is currently feasible with modern computing technology. It highlights the scalability issues of the label powerset method, which becomes impractical for problems with large numbers of class labels. In addition to this high RAM requirements, to get the full ranking we would have to evaluate 2^{983} classifiers which would take many CPU cycles for even a single data point.

B. ONE-VS-ALL (OVA) HD

One-vs-all is a problem transformation method used in multi-label classification where the problem is transformed into multiple binary classification problems. In this method, a separate binary classifier is trained for each label, where each classifier predicts whether the instance belongs to the corresponding label or not. Formally, given an instance $\mathbf{x}$ with L possible class labels, the one-vs-all method creates L separate binary class vectors $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_L$ of length 2 that represent the presence or absence of each class label. For each binary classification problem i, a binary label is assigned as follows:

$$\mathbf{y}_i = \begin{cases} 1 & \text{if } y_j = i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where y_j is the label vector of the instance $\mathbf{x}$. Given an instance $\mathbf{x}$ with L possible class labels, the OVA method creates L binary class vectors $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_L$, where $\mathbf{y}_i$ indicates whether the instance belongs to the i^{th} class or not. The i^{th} classifier is trained using the binary class vector $\mathbf{y}_i$ as the target variable, and the output of the i^{th} classifier is interpreted as the probability of the instance belonging to the i^{th} class.

The one-vs-all method is computationally efficient and scales well with the number of class labels, making it suitable for larger-scale multi-label classification problems. However, it suffers from the issue of label correlation as it treats each label independently, ignoring any correlations that may exist between them.

OvA HD approach involves the creation of two centroid hypervectors for each label, resulting in $2L$ labels. The two hypervectors for each class denote the positive and negative associations of that label. Together, the pair of hypervectors represent the binary classifier for a single label. During inference, we encode the test instance and evaluate it using our L binary classifiers, each of which predicts the relevance of its corresponding label. The predictions of all classifiers are then combined to give the final inferred label vector.

Compute Realization Cost of OvA HD: For the OvA HD approach, we need to create two centroid hypervectors for each label, resulting in $2L$ labels. For the example of Delicious dataset, there are 983 labels, so we need to create 1966 hypervectors in total. For a hypervector dimensionality of 1024, each hypervector will require 256 bytes of storage. Therefore, the total storage required for loading the HDC model can be calculated as 1966 hypervectors $\times 256$ bytes which is 491.5 kilobytes, which is significantly smaller than PowerSetHD.

The complexity analysis for classifying a single data point using HDC depends on the number of labels and the dimensionality of the hypervectors. Since we are using the OvA HD approach with 983 labels and a hypervector dimensionality of 1024, the time complexity for classifying a single data point can be expressed as $O(LD)$, where L is the number of labels and D is the hypervector dimensionality. In practice, the complexity may be higher due to the need to compute distances between the test instance and all hypervectors, as well as the need to combine the predictions of all binary classifiers. However, the OvA HD approach is computationally efficient and scales well with the number of class labels, making it more suitable for larger-scale multi-label classification problems compared to PowerSet HD.

V. TINYXML HD: EXTREME MULTI-LABEL CLASSIFICATION

Extreme multi-label classification (XMLC) [72], [73] represents a challenging variant of multi-label classification, where the task involves predicting a large number of labels for each instance in a dataset. The scale of the label space in XMLC can range from thousands to millions, making it extremely challenging for traditional multi-label classifiers to handle efficiently. This presents significant scalability and computational challenges, particularly compared to small multi-label classification problems, such as those described in the previous section, where the label set is relatively small. One specific variant of XMLC that has gained traction in various real-world applications, such as text categorization [41] and recommendation systems [74], is Extreme Multi-Label Text Classification (XMTC) [75]. The goal of XMTC is to classify documents into a potentially large number of labels. In the rest of this paper, we describe TinyXML HD, which solves XMTC problem by leveraging hyperdimensional representations.

A. HYPERVECTORS FOR TEXTUAL DATA

The XMTC datasets offer text data in two forms: bag-of-words representation or raw-text. The bag-of-words (BoW) is a widely used text representation approach in natural language processing (NLP) that represents a document as a collection of words with the frequency of their occurrences, disregarding the order of the words. In our TinyXML HD, if raw-text data is available, we leverage the Word2Vec [76], [77] embeddings for representing text; otherwise, we use bag-of-words. **TinyXML HD BoW encoding** projects BoW feature vector into a hypervector using Random Projection Encoding [25], as described in Section III.

Raw text data poses a challenge. A simple and meaningful strategy is to consider the compositional distributional semantics approach. Compositional distributional semantics is a method of representing the meaning of a sentence as a function of the meanings of its constituent words. This approach is based on the distributional hypothesis, which posits that words that appear in similar contexts tend to have similar meanings [78]. Given a sentence S consisting of n words, represented as d-dimensional vectors $w_1, w_2, ..., w_n$, we can combine these vectors using a composition function f to obtain a sentence vector s:

$$s = f(w_1, w_2, ..., w_n) \tag{3}$$

The composition function f takes the word vectors as input and returns a single vector representing the meaning of the sentence. There are various ways to define the composition function, such as averaging the word vectors and concatenating them [79], [80].

One approach for representing a sentence as a composition of words is to assign random symbolic hypervectors to each word in the dataset and then use compositional distributional semantics to obtain a sentence vector. Previous studies [1], [10] have employed this approach with varying degrees of success in various NLP tasks. However, a key issue with this approach is that it ignores the structural relationships between words. Models like word2vec [76], [77] address such issues by generating vector representations that capture semantic relationships between words in a meaningful way.

Word2Vec embeddings for TinyXML HD: We leverage Word2Vec with Hyperdimensional encoding for learning in TinyXML HD. Word2Vec is a powerful method for creating

distributed vector representations of words that capture semantic and syntactic aspects of natural language processing tasks [76], [77]. Traditional approaches to representing words rely heavily on sparse one-hot vector representations, which are high-dimensional and lack the ability to capture the subtle nuances of word meanings. In contrast, Word2Vec's distributed vector representations encode semantic relationships between words by placing words with similar meanings closer together in the vector space [76], [77].

To encode an instance in TinyXML HD, we first obtain the Word2Vec embeddings of all its constituent words. We then project these embeddings into hypervector using Random Projection encoding described in Section III. Finally, to employ compositional distributional semantics, we *bundle* ($\oplus$) the resultant hypervectors into a single hypervector that represents an instance. As mentioned in Section III, Random Projection Encoding preserves the euclidean distance, so this enables the generation of symbolic hypervectors that capture semantic information between words through their cosine similarity scores, resulting in expressive high-dimensional representations. Consequently, hypervectors for words that are similar will be proportionally similar and those of semantically dissimilar words would be dissimilar. In this way we are able to capture the complex relationships between words and obtain a richer representation that conveys more information about their semantic context.

TinyXML HD Label Representation: We leverage HDC algebra to map and combine multiple labels in hyperdimensional space. Let L be the number of labels or symbols in the dataset, and $\mathcal{H}^D$ be a D-dimensional hyperdimensional space with $H = \{+1, -1\}$ for the MAP HDC model. We map each label to a hypervector in this space. The initialization of the label space $\mathcal{Y}_{1...L}$ involves assigning a random hypervector from a Binomial distribution to each label. Specifically, we initialize each label $\mathcal{Y}_i$ by sampling from $\mathcal{B}(0.5) \cdot 2 - 1$. To obtain a high-dimensional representation of a label, we bundle the corresponding hypervectors of the labels present for an instance, denoted by $\mathcal{Y}^p$. We then combine these hypervectors using the hyperdimensional operator $\oplus$ to obtain a single hypervector representation for the instance $\mathbf{x}_i$, given by Eq. (V-A). This operator is commutative, which allows us to bundle the hypervectors in any order without affecting the final result. $\mathbf{y}_i = \bigoplus_{j \in \mathcal{Y}^p} \mathcal{Y}_j$

B. LEARNING WITH TINYXML HD

With both the inputs and outputs embedded in the same high-dimensional space, the next step is to learn a mapping $f : \mathbf{x_i} \in \mathcal{H} \rightarrow \mathbf{y_i} \in \mathcal{H}$, where $\mathbf{x_i}$ is the input hypervector and f outputs the hypervector that represents all the labels present for that instance. In this section, we present our proposed neural network based approach to learn this mapping function f. Our proposed approach presents a linear formulation over the HDC operators of *binding* and *bundling*, which allows for effective and efficient optimization using gradient-based methods.

Objective formulation: We decompose the original learning problem into multiple sub-problems to enhance its learnability. Let's consider an example where we break down the learning problem into two sub-problems, which we rewrite as

$$f : \mathbf{H_1} \in \mathcal{H}^D \rightarrow \mathbf{H_2} \in \mathcal{H}^D \tag{4}$$
$$f = f_1(f_2) \tag{5}$$
$$f_1 : \mathbf{H_1} \in \mathcal{H}^D \rightarrow \mathbf{H}_x \in \mathcal{H}^D \tag{6}$$
$$f_2 : \mathbf{H}_x \in \mathcal{H}^D \rightarrow \mathbf{H_2} \in \mathcal{H}^D \tag{7}$$

where f_1 maps the input instance to an intermediate hypervector $\mathbf{H}_x$, and f_2 maps $\mathbf{H}_x$ to the output label hypervector $\mathbf{H_2}$.

We define f_1 and f_2 using the HDC arithmetic operations of *binding* and *bundling*. In particular, we parameterize f_1 as

$$f_1 = \mathbf{H_1} \otimes H_{\text{conv1}} \tag{8}$$

where H_{conv} is a hypervector to be learned. Similarly, we define f_2 as

$$f_2 = \mathbf{H_x} \otimes H_{\text{conv2}} \tag{9}$$

This approach considers the mapping between two hypervectors as a series of geometric transformations where the input hypervector is bound sequentially with the intermediate hypervectors induced by the sub-problems. The hyperparameters of the number of sub-problems to induce and the dimensions of the learned hypervectors are chosen based on the complexity of the dataset.

1D Convolutions as Hypervector Operators: In developing a neural architecture, it is crucial to adhere to the principles of high-dimensional computing (HDC), which dictate that representations should be distributed, with individual coordinates devoid of semantic information. Consequently, our neural architecture should interpret inputs as distributed representations rather than feature vectors containing discrete semantic entities. To achieve this, we use one-dimensional convolutional operators as hypervector operators. A one-dimensional convolution involves applying a filter across an input, using a single set of weights to process the entire hypervector. This operation treats each vector region independently with the filter, synthesizing a vector that encapsulates information from the input. In contrast, a fully connected (FC) network utilizes an interconnected network of connections to process all coordinates of the input vector, treating the coordinates as dependent entities, which contravenes the principles of HDC representations.

Figure 1 details the architecture and operations of our proposed ConvHD block. Our ConvHD block consists of three layers parameterized by C, which represents the expansion factor and F, the filter size. The block consists of three convolutional layers: the first layer (X_1) takes input hypervectors and generates C hypervectors, the second layer (X_2) processes these C hypervectors to produce $C/2$ hypervectors, and the third layer (X_3) combines these $C/2$ vectors

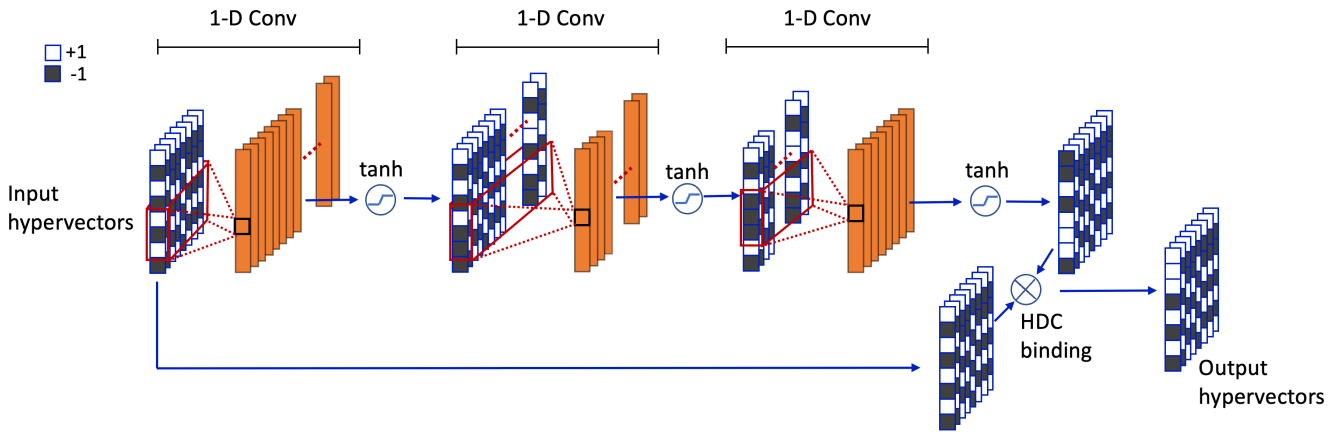


FIGURE 1: ConvHD Operator

into a single hypervector. A single convolutional unit can be defined as follows:

$$g : \mathcal{H}^D \rightarrow \mathcal{H}^D \tag{10}$$

$$g = X_3 \circledast \tanh\left(X_2 \circledast \tanh\left(\tanh\left(X_1 \circledast H_{\text{input}}\right)\right)\right) \tag{11}$$

The ConvHD block is be represented by:

$$\text{ConvHD} = g(H_{\text{input}}) \otimes H_{\text{input}} \tag{12}$$

Hence a model f using 2 ConvHD blocks is represented by:

$$f = \mathcal{H}^D \rightarrow \mathcal{H}^D \tag{13}$$

$$f = \text{ConvHD}\left(\text{ConvHD}(H_{\text{input}})\right) \tag{14}$$

We formulate the sub-problems as a single learning problem, where we optimize the parameters X_1, X_2, X_3 using gradient-based methods. To enhance our architecture, we incorporate the idea of dilated convolutions [81] to increase the receptive field of the convolutional layers [82]. We also set the filter size F to be large, approximately a quarter of the hypervector dimensionality D. These details are crucial, as they increase the effective receptive field with every 1-D convolution operation. That is, they increase the number of hypervector coordinates in the input that influence the synthesis of a single coordinate in the output of the last convolution layer. By using a large filter size, we increase the number of coordinates in the input hypervector that are considered to produce a single coordinate in the resultant hypervector. Similarly, dilation helps to increase the receptive field by allowing deeper layers to infer coordinates based on a larger area of the input hypervector. Since the ConvHD operator uses three 1-D convolutional layers, the receptive field increases progressively with each layer looking at a larger section of the input hypervector to make a decision.

The expansion factor C spawns more sub-problems parallely. For instance, in the above example of breaking down the learning problem into 2 parts, if we set $C = 2$, then each layer estimates 2 sets of sub-problems. The first layer will parallely solve two sub-problems similar to Equation 8 and similarly the second layer will solve two sub-problems similar to Equation 9. The results of the 2 sub-problems will then be combined through the bundling operation.

In order to learn the mapping to solve these sub-problems, we use the loss function detailed in [18], which aims to minimize the cosine distance between the predicted hypervector and the ground-truth label hypervector.

VI. EVALUATION OF OVA, POWERSET & TINYXML HD

This section of our research paper presents the results of our proposed multi-label classification approach on various real-world datasets. Through a series of experiments, we demonstrate the trade-off between compute efficiency and accuracy of our approach across a range of complexity levels, from small-scale (less than 20 labels) to extreme-scale (greater than 5000 labels). Our findings indicate that, in low-complexity scenarios with datasets of low cardinality, the One-vs-All HDC approach achieves high accuracy and efficiency. Conversely, the PowerSet HDC approach provides poor trade-offs, yielding benefits only when the label cardinality is very low, with efficiency degrading exponentially as complexity increases.

We evaluate the effectiveness of our proposed approach, TinyXML HD, on extreme size datasets. Our experiments demonstrate that TinyXML HD produces models that are 231x-836x smaller than state-of-the-art models while still achieving reasonable accuracy. Furthermore, our approach can efficiently train on large text datasets in just a few hours providing a speed up of up to 16x. These results highlight the potential of our proposed approach for solving extreme-scale multi-label classification problems while greatly reducing the computational resources required.

We evaluate the small scale problems on an Intel Xeon 24-core CPU while for the larger datasets we use a single Nvidia V100 GPU.

A. EVALUATION OF OVA & POWERSET HD

OvA and PowerSet HD Experimental Setup: We tested our OvA HD and PowerSet HD multi-label methods on

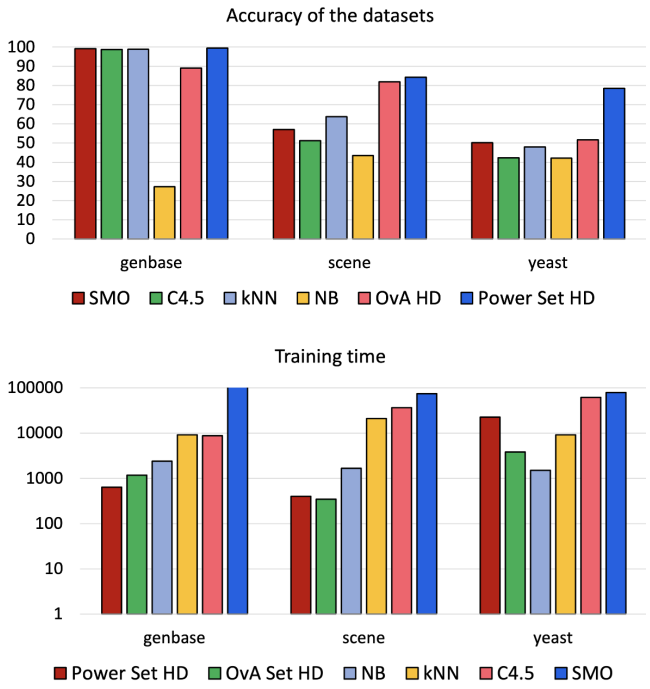


FIGURE 2: Efficiency of training

smaller size datasets by running on an optimized C++ implementation on an Intel Xeon 24-core CPU. We compare our HDC-based methods with multi-label versions of k-nearest neighbors (kNN) [48], Sequential Minimal Optimization – SMO [83], C4.5 [84], and Naive Bayes – NB [48], all of which are appropriate for smaller datasets. We utilized Java-based open-source Mulan [85] multi-label package with 3 small datasets for comparison:

Genbase [86] contains protein classes of 27 most important protein families, with 662 samples, each with 1186 attributes.

Scene [87] contains images with their characteristics and classes. One image can belong to up to 6 categories. It has 2407 samples, each with 294 attributes.

Yeast [88] has information about a set of yeast cells. The task is to determine the localization site of each cell amongst 14 possible sites. It has 2417 samples, each with 103 attributes.

OvA and PowerSet HD Accuracy: Figure 2 shows that OvA and PowerSet HD achieve comparable accuracy to state-of-the-art multi-label classifiers. PowerSet HD consistently outperforms state-of-the-art methods on all three datasets. OvA HD is slightly less accurate on the Genbase dataset but performs better on the Scene and Yeast datasets, likely due to their better separability of HD space compared to low-dimensional space.

OvA and PowerSet HD Performance & Efficiency: While PowerSet HD achieves higher accuracy, Figure 2 demonstrates that this comes at a significant cost in terms of execution time. This is due to the exponential increase in class hypervectors as discussed earlier. Figure 2 also shows that both OvA and PowerSet HD training are significantly faster than most other multi-label classifiers, with OvA HD being 60.8 times faster on average. PowerSet HD is only 3.5 times slower than OvA HD on datasets with a large portion of label combinations. Power Set HD is 24 times faster than state-of-the-art multi-label classifiers on average, or approximately two times slower than OvA HD, but offers 13% higher accuracy. For small datasets, where only a small subset of possible label combinations appear in the dataset, PowerSet HD can potentially be more efficient and accurate. However, for datasets with more number of possible label combinations, OvA HD is the clear choice as it offers a trade-off between compute efficiency and accuracy compared to PowerSet HD. These results indicate that the OvA HD approach is an ideal candidate for small scale multi-label classification tasks.

B. EXPERIMENTAL SETUP FOR TINYXML HD

We evaluated TinyXML HD HD on real-world, large-scale datasets from Extreme Multi-Label Text Classification (XMTC). Our objective is to maximize the compute efficiency of learning while achieving comparable precision to the state-of-the-art. For the XMTC dataset, we evaluate our proposed TinyXML HD on Nvidia V100 GPU.

Evaluation metrics: We consider Precision@k with $k = 1, 3, 5$ as our metric for evaluating the performance of TinyXML HD on multi-label classification, where k represents the top k predictions. This is a widely accepted and used evaluation metric by other works in literature [60], [61]. In addition, we evaluate the computational efficiency of TinyXML HD against the following start-of-the-art models: XT [89], Bonsai [90], SLEEC [50] and Parabel [91] for BoW datasets. For Raw text datasets, we consider these SoA models: AttentionXML [62], LightXML [61] and X-Transformer [60]. Given that previous research has not given a comprehensive account of the compute cost associated with these models, it is difficult to establish a standardized metric for comparison. To address this issue, we have considered two distinct metrics: the count of trainable parameters and the training time. The former serves as an indicator of the cost of training, since a model with a higher parameter count requires more gradients to be calculated and optimized, and is also indicative of greater model size. The latter is a direct measure of the time required to train the model. These two metrics offer a meaningful evaluation of compute cost in the context of real-world applications.

Datasets: In order to evaluate the expressiveness of our high-dimensional representations of text data, we select six datasets from Extreme Multi-Label Text Classification (XMTC) dataset, a widely accepted benchmark in literature [18], [72], [75], [98]. The datasets are described in Table 1. In addition to the scalability and computational challenges, the XMTC dataset poses an additional challenge which is the label sparsity issue. Bhatia et al [72] divided the datasets according to the number of labels per sample into small scale and large scale. Small scale datasets contain at most 5000 labels. Although pre-processed BoW features are available

TABLE 1: Dataset Metadata

Dataset	Feature type	# Labels	# Training points	# Testing points	Avg. Points per Label
Mediamill [92]	BoW	101	30,993	12,914	1902.15
Bibtex [93]	BoW	159	4,880	2,515	111.71
Delicious [94]	BoW	983	12,920	3,185	311.61
Eurlex-4K [95]	Text	3,993	5,000	3,993	25.73
Wiki10-31K [96]	BoW & Text	30,938	14,146	6,616	8.52
Amazon-13k [97]	Text	13,330	1,186,239	306,782	448.57

for all datasets, the original text is not. Consequently, we use the original text when available and BoW for all others.

TinyXML HD HD Architecture Specifics: We use Random Projection Encoding as discussed in Sec. III for BoW feature representations, while for raw text datasets, we utilize the combination of Random Projection Encoding with Word2Vec as described in Sec. V. We employ ConvHD blocks with expansion factor $C = 128$, filter size $F = 255$ with dilation set to 7 and a hypervector dimensionality of 1024. To optimize the model, we use the loss function proposed by Ganesan et al. [18] but we remove the negative loss component, which was intended to ensure that the output hypervector from the model $f(.)$ is orthogonal to the labels that are not present for that instance. Since all labels are initialized with random hypervectors that are orthogonal to each other, enforcing the similarity to the present labels alone will automatically satisfy the orthogonality condition with the labels not present. Therefore, we only retain the positive component in the loss function, and we discard the additional positive $\mathbf{p}$ vector used in [18] as it does not improve results. The final loss function is as follows: $\mathcal{L} = \sum_{c^p \in \mathcal{Y}^p}(1 - \cos(\mathbf{y_i}, c^p))$ where $\mathbf{y_i}$ is the final hypervector output by our model $f(\mathbf{x_i})$ for the i-th instance, and c^p represents a present label. The loss function aims to minimize the cosine distance between $\mathbf{y_i}$ and all present labels, thereby encouraging the model to produce a hypervector that is more similar to the labels that are present in the instance.

Comparison baselines: Our evaluation comprises two parts, with datasets divided by the type of features used. We consider different baselines for each part. For BoW datasets, we benchmark against other state-of-the-art models that use the same features, such as Bonsai [90], Parabel [91], and PFastreXML [99]. For raw-text datasets, we compare TinyXML HD's performance against state-of-the-art deep learning approaches, including AttentionXML [62], X-Transformer [60], and Light-XML [61]. These deep learning models employ powerful architectures like transformers, with hundreds of millions of parameters, enabling them to extract highly expressive embeddings from text data. As a result, TinyXML HD is inherently disadvantaged due to the significant disparity in parameter count. The primary objective of this research is to optimize size, speed and accuracy tradeoffs of such constrained HDC models to evaluate their viability as a lightweight paradigm. Hence, we aim to achieve reasonable accuracy with respect to the state-of-the-art, within a 10% margin.

C. EVALUATION OF TINYXML HD HD

TinyXML HD, PowerSet & OvA HD Comparison: To gain insight into the trade-off between performance and accuracy, we have evaluated TinyXML HD on small-scale multi-label classification tasks. In this study, we compare the performance of TinyXML HD to that of PowerSet and OvA HD on three small datasets, as described in Section VI-A. Given the lower complexity of the task at hand, we have scaled down TinyXML HD by utilizing a depth of 1, a block size of 8, and a filter size of 255. As the datasets used have low cardinality, we have evaluated our approaches using overall accuracy since the precision@K metrics are inapplicable for $K = 3, 5$, due to the limited number of labels per instance. In addition, considering the low complexity of the task, we evaluate performance only on CPU and do not use any specialized hardware for acceleration.

Our results in Table 2 show that TinyXML HD achieves 100% accuracy on Genbase [86], whereas the performance drops by 8% on Scene [87] and 3% on Yeast [88]. The most likely reason for the lower accuracy on the two datasets is the scarcity of training data. Problem transformation techniques were trained faster than TinyXML HD, despite the latter's smaller size. The only exception to this was the Yeast [88] dataset, on which TinyXML HD was significantly faster (1.2x over OvA HD and 7.4x over Power Set HD). This is due to the disparity in label cardinality across the datasets. Genbase [86] and Scene [87] have label cardinalities of 1.25 and 1.07, respectively, meaning that only a single centroid vector needs to be updated for Power Set HD and OvA HD. However, the Yeast [88] dataset has a label cardinality of 4.2, requiring OvA HD to update 4 centroid hypervectors while Power Set HD needs to update $2^4 = 16$ centroid hypervectors for each instance, resulting in increased training time.

The higher training time of TinyXML HD can be attributed to the convolution operations, which are computationally intensive compared to the HDC operations of bundling, binding, and similarity check used by the transformation methods. These operations reduce to simple element-wise additions and multiplications, making them easier to compute. Consequently, Power Set HD and OvA HD can be easily parallelized and accelerated in hardware, while the convolution operation of TinyXML HD would be harder to accelerate.

The dissimilarity in parameter count between the models is attributed to their respective architectures. The parameter count for PowerSet HD increases exponentially with the size

TABLE 2: TinyXML HD on Small Scale Multi-Label Classification (normalized to TinyXML HD)

Model	Genbase [86]			Scene [87]			Yeast [88]		
	Accuracy	# params	Training time	Accuracy	# params	Training time	Accuracy	# params	Training time
TinyXML HD	**100**	**1**	1	76.0	1	1	75.5	1	**1**
PowerSet HD	99.5	30.4K	**0.54**	**84.2**	1.99	0.14	**78.4**	510.6	7.4
OvA HD	89.1	1.677	0.6	81.9	**0.03**	**0.12**	51.7	**0.869**	1.26

of the label set. Conversely, the parameter count for the OvA HD scales linearly with the label set size, resulting in a parameter count that is twice the size of the label set for our implementation of the one-vs-all classifier. In contrast, TinyXML HD leverages only one hypervector to represent each label, with the additional parameters solely corresponding to convolution filters. These parameters are minimal in comparison to the label set size, further underscoring the efficiency of the TinyXML HD architecture.

The current study has revealed that the HDC-based problem transformation approaches offer a significantly superior trade-off between training time and accuracy for small-scale multi-label classification tasks compared to TinyXML HD. Specifically, for datasets where only a limited subset of possible label combinations appear in the dataset, PowerSet HD exhibits the potential to be both more efficient and accurate. In contrast, for datasets with a larger number of possible label combinations, albeit less than at the extreme scale, OvA HD proves to be a more promising candidate. While TinyXML HD boasts a smaller parameter count, the parameter count of OvA HD remains comparable and is sufficiently small for the complexity scale under investigation.

Due to linear and exponential scaling of PowerSet HD and OvA HD, these methods are unsuitable for extreme-scale multi-label classification tasks. PowerSet HD is too large to implement, while OvA HD takes too long to train. We next evaluate TinyXML HD on extreme-scale multi-label classification.

TinyXML HD Multi-label Accuracy: We investigate the performance of TinyXML HD on extreme scale datasets next. Table 3 presents the performance of TinyXML HD along with its respective baselines on the BoW dataset. Our findings reveal that for Mediamill and Wiki10-31K BoW, TinyXML HD's precision at top one (p@1) is within 5% of the state-of-the-art (SoA). However, we note that barring Mediamill, precision at top three (p@3) and precision at top five (p@5) is lower across all datasets when compared to the SoA.

Table 4 shows the performance of TinyXML HD and its respective baselines on the raw text datasets. Our findings reveal that TinyXML HD's precision is relatively lower for these datasets. While we observe that TinyXML HD achieves comparable performance on the Wiki10-31K dataset, the precision drops for Amazon-13k by 9%. As we attempt to retrieve more labels from the hypervector, the retrieval becomes less robust. Considering that Wiki10-31K has 31,000 labels with only 8 samples per label available, the performance of TinyXML HD (83%) is remarkable. While the performance of TinyXML HD is not extraordinary compared to the state-of-the-art, it is remarkable considering the fact that TinyXML HD relies on a simple encoding scheme. In contrast, the state-of-the-art models employ highly complex architectures with millions of parameters. This observation validates the potential of HDC to provide expressive representations of data at extraordinarily low compute costs. Moreover, unlike other models where the model size scales almost linearly with label size, TinyXML HD ensures that the output size of the model is fixed to the dimensionality of the hypervector D independent of label set size.

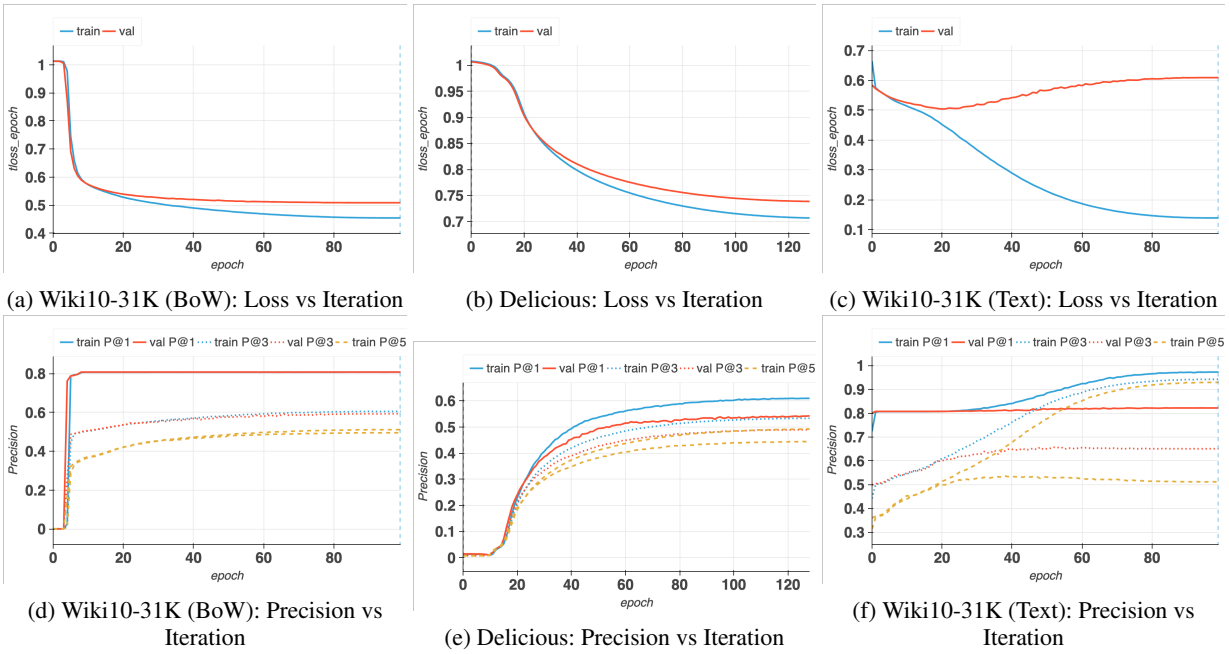
TinyXML HD Convergence: Figure 3 showcases the convergence plots of TinyXML HD on three distinct datasets: Wiki10-31K (BoW) [96], Delicious [94], and Wiki10-31K (Text) [96]. When examining the BoW datasets, namely Wiki10-31K [96] and Delicious [94], a notable trend emerges. The loss function exhibits a smooth decrease, punctuated by a slight, yet discernible, initial drop for Wiki10-31K. In stark contrast, the Wiki10-31K (Text) variant converges in fewer than 30 epochs and seemingly starts to overfit, but, the precision plots provide further insights into this phenomenon. While P@1 and P@3 seem to have converged, a closer analysis reveals that P@5 continues to improve. This observation suggests that the model is still assimilating new information from the data. Although unable to enhance P@1 and P@3 further, the model's focus shifts to effectively ranking two additional labels.

TinyXML HD ROC and AUC For the readers' benefit, we also provide additional insights in the form of Receiver Operator Characteristics (ROC) Curves in Fig. 4a and the corresponding Area Under the Curve (AUC) values as shown in Fig.4b. Although these metrics are typically employed for evaluating multi-class classifiers, considering mutually independent labels, we adapt them to our multi-label setting by treating each label as an independent binary classification problem. Due to computational constraints, we focus on the Delicious dataset for these metrics, as it contains a large number of labels. To ensure plot coherence, we present the ROC curve for 10 randomly selected labels. Furthermore, in order to comprehend the AUC, we visualize the distribution of AUC values obtained for each label across the dataset's 983 labels. These results show expected accuracy when treating multi-label problem as an independent binary classification problem.

TinyXML HD Overfitting To address the issue of overfitting, we explored the utilization of BatchNorm-2D [104], L1/L2 Regularization, and Dropout techniques [105]. How-

TABLE 3: Multi-Label Classification Performance on BoW datasets: Comparison with State-of-the-Art

Dataset		TinyXML HD	FastXML [100]	Parabel [101]	PfastreXML [102]	SLEEC [50]
Medialmill [92]	p@1	82.1	83.5	Not Reported	84.2	84.0
	p@3	64.4	65.7		67.3	67.2
	p@5	50.0	49.9		53.0	52.8
Delicious [94]	p@1	62.7	69.6	67.4	67.1	67.5
	p@3	55.7	64.1	61.8	62.3	61.3
	p@5	51.4	59.2	56.7	58.6	56.5
Wiki10-31K (BoW) [96]	p@1	80.8	83.0	84.1	83.5	85.8
	p@3	50.5	67.47	72.4	68.6	72.9
	p@5	44.3	57.7	63.3	59.1	62.7



(a) Wiki10-31K (BoW): Loss vs Iteration

(b) Delicious: Loss vs Iteration

(c) Wiki10-31K (Text): Loss vs Iteration

(d) Wiki10-31K (BoW): Precision vs Iteration

(e) Delicious: Precision vs Iteration

(f) Wiki10-31K (Text): Precision vs Iteration

FIGURE 3: Loss and Precision vs Iteration for Three Datasets

ever, neither of these approaches proved to be effective. Additionally, we conducted experiments to further reduce the parameter count, but since the model already consisted of only 2.5M parameters, any additional reduction led to a decrease in accuracy.

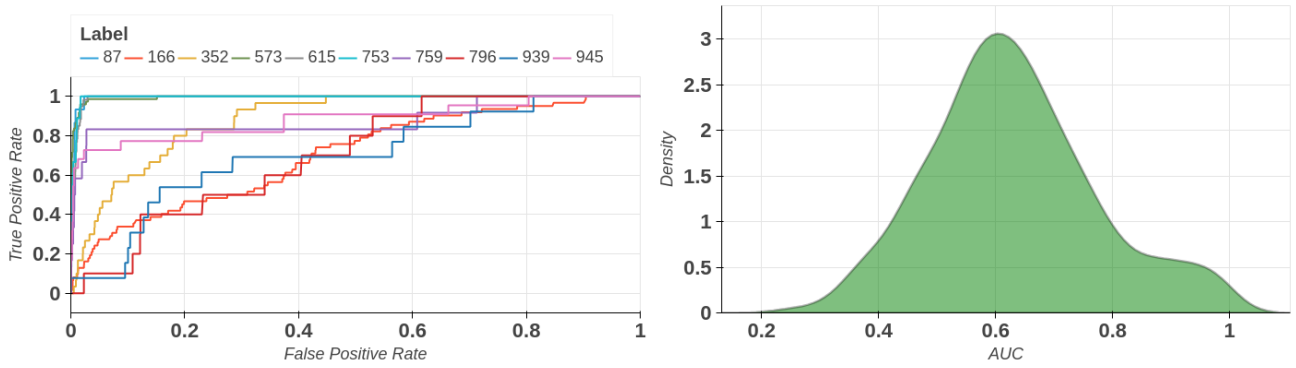
TinyXML HD Robustness To briefly examine the ability of the method to perform when few labels are missing, for every sample we dropped one label with probability p. Our experiments show that up to $p = 0.2$ there is no accuracy degradation, showing robustness of TinyXML HD.

TinyXML HD Computing Efficiency: We compare the efficiency of TinyXML HD to the following state-of-the-art models: XT [89], Bonsai [90], SLEEC [50] and Parabel [91]. Table 6 compares the training time and model size of TinyXML HD against the state-of-the art listed above on the Wiki10-31K BoW dataset. Remarkably, TinyXML HD trains in 10 mins with a minuscule model size of 19.8MB while achieving comparable precision on the dataset. We observe that TinyXML HD is 6.5x smaller than the smallest SoA (Bonsai [90]) and 56x smaller than the largest SoA (SLEEC [50]). Methods such as Parabel [91], Bonsai [90] build multiple probabilistic label trees and perform classification on each node which becomes computationally expensive very quickly. Consequently, TinyXML HD is 1.25x quicker than the fastest SoA (Parabel [91]) and 4x quicker than the slowest SoA (Bonsai [90]).

Raw text training time and the number of parameters needed for Amazon-670K dataset is shown in Table 5. All the deep learning models necessitate several days to train on this extensive dataset. In stark contrast, TinyXML HD showcases a remarkable training speed of merely six hours, even though the dataset has over 670K labels and 130K training samples. TinyXML HD provides a speedup of 4x over the fastest SoA (AttentionXML [62]) and 16x over the slowest SoA (X-Transformer [60]) This exceptional speedup can be attributed to two crucial factors. First, the deep learning models rely on complex transformer models like BERT and RoBERTa, to extract highly expressive feature embeddings from data. In contrast, TinyXML HD employs a simple encoding scheme, that decomposes into highly parallelizeable operations. The

TABLE 4: Multi-Label Classificaiton Performance on Real Text Datasets: Comparison with State-of-the-art

Dataset		TinyXML HD	AttentioinXML [62]	XTransformer [103]	LightXML [98]
Eurlex-4K [95]	p@1	61.3	87.1	87.2	87.63
	p@3	51.8	73.9	75.1	75.89
	p@5	43.7	61.9	62.9	63.36
Wiki10-31K [96]	p@1	83.3	87.4	88.5	89.5
	p@3	66.2	78.4	78.7	78.9
	p@5	60.7	69.3	69.6	69.8
Amazon-13K [97]	p@1	86.2	95.9	96.7	96.7
	p@3	60.4	82.4	83.8	84.0
	p@5	44.6	67.3	68.5	68.7



(a) ROC Curves for 10 different labels

(b) Probability distribution of AUC values of all Delicious labels

FIGURE 4: (Left) ROC plots for 10 different labels (Right) Density of AUC values across all labels from the Delicious dataset considering each label as a binary classification problem

TABLE 5: Raw text model size & training time

Model	Model size	Training time
AttentionXML [62]	16.56 GB	26.30 hrs
Xtransformer [60]	>5GB	> 35 hrs [61]
LightXML [61]	4.59GB	28.75 hrs
TinyXML HD	**19.8MB**	**6 hours**

TABLE 6: BoW model compute efficiency

Model	Train time	Size
Parabel [91]	0.20 hrs	180MB
SLEEC [50]	0.21 hrs	1.13GB
Bonsai [90]	0.64 hrs	130MB
XT [89]	0.39 hrs	370MB
TinyXML HD	**0.16hrs**	**19.8MB**

bulky feature extractor is replaced by our lightweight HDC-based encoding, which demonstrates the expressiveness of these representations when used to encode relevant features. Second, the output dimensionality of deep learning models typically scales with the label set size (L). However, TinyXML HD ensures that the output size of the model is fixed to the dimensionality of the hypervector (D), where $D << L$, irrespective of the label size. This unique feature allows for the reduction of the number of trainable parameters, thereby improving training efficiency and reducing the computational load.

These results clearly demonstrate the strength of HDC when it comes to computational cost of learning. HDC has enormous potential to make learning computations tractable and to dramatically cut down on training time with good accuracy. TinyXML HD is 836x smaller than the largest SoA (AttentionXML [62]) and 231x smaller than the smallest SoA (LightXML [61]). Considering that X-Transformer [60] uses an ensemble of transformers we suspect that X-Transformer would be larger than 5GB and would require 100 hours of effort to train [61] making it infeasible to compare with.

VII. CONCLUSION

In this work, we have presented novel approaches to Multi-Label classification using Hyperdimensional Computing (HDC), addressing the entire spectrum of complexity. For small scale Multi-Label classification, we proposed using HDC to implement two problem transformation methods: PowerSet transform and One-vs-All transform. Through rigorous evaluation, we demonstrated that in low complexity scenarios, OvA HD can provide up to 60x speedup in low cardinality datasets, while PowerSet HD can be up to 24x faster than SoA with comparable accuracy on datasets where few labels occur together, especially in low cardinality datasets. For the extreme multi-label classification problem, where label size is very large, we proposed a neuro-symbolic approach, TinyXML HD, that breaks down the learning

problem into multiple sub-problems using hyperdimensional arithmetic and then uses gradient optimization to solve these sub-problems. Our results demonstrated that TinyXML HD can dramatically compute the computational complexity of multi-label learning on large-scale real-world datasets while achieving good accuarcy. TinyXML is 836x smaller than the largest SoA and 231x smaller than the smallest for text datasets and up to 16x faster to train. Similarly, for BoW datasets, TinyXML is 6.5x - 56x smaller than SoA models while training being up to 4x quicker.

REFERENCES

[1] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in Proceedings of the 22nd Annual Conference of the Cognitive Science Society, 2000, pp. 103–108.

[2] ——, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," Cognitive Computation, vol. 1, no. 2, pp. 139–159, 2008.

[3] B. Khaleghi, J. Kang, H. Xu, J. Morris, and T. Rosing, "Generic: Highly efficient learning engine on edge using hyperdimensional computing," in Design Automation Conference (DAC), 2022.

[4] B. Khaleghi, M. Imani, and T. Rosing, "Prive-hd: Privacy-preserved hyperdimensional computing," in IEEE/ACM Design Automation Conference (DAC). IEEE, 2020.

[5] B. Khaleghi, H. Xu, J. Morris, and T. Rosing, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, 2021, pp. 1–6.

[6] A. Dutta, S. Gupta, B. Khaleghi, R. Chandrasekaran, W. Xu, and T. Rosing, "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction," in Proceedings of the 32nd edition of the Great Lakes Symposium on VLSI. ACM, 2022, pp. 149–154.

[7] Y. Guo, M. Imani, J. Kang, S. Salamat, J. Morris, B. Aksanli, Y. Kim, and T. Rosing, "Hyperrec: Efficient recommender systems with hyperdimensional computing," in 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2021, pp. 118–123.

[8] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting hyperdimensional learning for fpga and low-power architectures," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021.

[9] R. W. Gayler, "Multiplicative binding, representation operators & analogy," in Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences, 1998, pp. 1–4.

[10] ——, "A vector symbolic architecture for composing and retrieving predicate-argument relations," Connection Science, vol. 15, no. 2, pp. 119–140, 2003.

[11] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 1224–1229.

[12] S. Salamat, J. Kang, Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Fpga acceleration of protein back-translation and alignment," in Design, Automation and Test in Europe Conference and Exhibition (DATE). IEEE, 2021.

[13] A. Paleo, K. D. Carlson, and L. S. Davis, "Hyperdimensional object representation for scene context characterization," in IEEE Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2009, pp. 79–86.

[14] R. W. Gayler, "Vector symbolic architectures: a new direction in artificial intelligence," in AAAI/IAAI. AAAI Press/The MIT Press, 2002, pp. 959–966.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[16] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, 2016.

[17] T. A. Plate, "Holographic reduced representations," IEEE Transactions on Neural Networks, vol. 6, no. 3, pp. 623–641, 1995.

[18] A. Ganesan, H. Gao, S. Gandhi, E. Raff, T. Oates, J. Holt, and M. McLean, "Learning with holographic reduced representations," Advances in Neural Information Processing Systems, vol. 34, pp. 25 606–25 620, 2021.

[19] G. Hinton, "Learning multiple layers of representation," Trends in cognitive sciences, vol. 11, no. 10, pp. 428–434, 2007.

[20] Y. Bengio, "Learning deep architectures for ai," Foundations and Trends® in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.

[21] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.

[22] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," Cognitive Computation, vol. 1, no. 2, pp. 139–159, 2009.

[23] S. I. Gallant, J. A. Rodriguez, and B. J. Gallant, "Toward a human-like open-domain chatbot," Brain Sciences, vol. 3, no. 1, pp. 1–49, 2013.

[24] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 12, pp. 2691–2706, 2016.

[25] A. Thomas, S. Dasgupta, and T. Rosing, "Theoretical foundations of hyperdimensional computing," Foundations and Trends in Electronic Design Automation, vol. 11, no. 1, pp. 1–117, 2017.

[26] L. Lai and M. K. Sundareshan, "Hdc: A high-dimensional classifier," Pattern Recognition, vol. 60, pp. 25–39, 2016.

[27] M. Imani and T. S. Rosing, "Hdc: A high-dimensional computing framework for approximate regression," IEEE Transactions on Computers, vol. 68, no. 12, pp. 1799–1813, 2019.

[28] A. Goudarzi, M. Imani, and T. S. Rosing, "Hyperdimensional computing for reinforcement learning," in Proceedings of the 38th International Conference on Machine Learning. PMLR, 2021, pp. 3832–3841.

[29] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. S. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in IEEE/ACM Design Automation and Test in Europe Conference (DATE), 2019, pp. 838–841.

[30] F. Asgarinejad, A. Thomas, and T. Rosing, "Detection of epileptic seizures from surface eeg using hyperdimensional computing," in 2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC). IEEE, 2020, pp. 2243–2246.

[31] V. Vapnik and A. Chervonenkis, "A new approach to the problem of pattern recognition based on learning and decision functions of many variables," Automation and Remote Control, vol. 24, no. 5, pp. 774–780, 1963.

[32] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[33] E. P. Frady and F. T. Sommer, "Extending hyperdimensional computing with gradient information," in 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2018, pp. 1–5.

[34] ——, "Gradient-based learning in high-dimensional random neural networks," Neural Networks, vol. 124, pp. 304–311, 2020.

[35] Y. Wang, M. Lin, J. Wang, J. Zhang, and J. Zhou, "Gradient-based hyperdimensional computing for clustering and classification," IEEE Access, vol. 8, pp. 39 277–39 287, 2020.

[36] Q. Su, J. Yang, and X. Wu, "Gradient-based hyperdimensional computing for deep learning," IEEE Access, vol. 9, pp. 36 736–36 746, 2021.

[37] J. Zhou, Y. Wang, M. Lin, J. Zhang, and J. Yang, "Unsupervised gradient-based hyperdimensional computing," IEEE Access, vol. 9, pp. 16 555–16 565, 2021.

[38] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[39] Q. Liao and X. Yuan, "Replace or retrieve keywords in document for information retrieval," arXiv preprint arXiv:1905.01823, 2019.

[40] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative long short-term memory," arXiv preprint arXiv:1602.03032, 2016.

[41] G. Tsoumakas and I. Vlahavas, "An introduction to multi-label classification," International Journal of Data Warehousing and Mining, vol. 3, no. 3, pp. 1–13, 2007.

[42] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 8, pp. 1819–1837, 2014.

[43] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence. IOS Press, 2010, pp. 1007–1012.

[44] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Dzeroski, "A literature survey on algorithms for multi-label learning," arXiv preprint arXiv:1502.05988, 2015.

[45] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," International Journal of Data Warehousing and Mining, vol. 3, no. 3, pp. 1–13, 2007.

[46] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," IEEE transactions on knowledge and data engineering, vol. 23, no. 7, pp. 1079–1089, 2011.

[47] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," Machine learning, vol. 85, no. 3, pp. 333–359, 2011.

[48] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," Pattern Recognition, vol. 40, no. 7, pp. 2038–2048, 2007.

[49] E. L. Mencia and M. N. Moreno, "Multi-label decision trees," Pattern Analysis and Applications, vol. 11, no. 1, pp. 43–52, 2008.

[50] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain, "Sparse local embeddings for extreme multi-label classification," in Advances in neural information processing systems, 2015, pp. 730–738.

[51] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in CVPR 2011. IEEE, 2011, pp. 817–824.

[52] S. Mostafavi, D. Ray, D. Warde-Farley, C. Grouios, and Q. Morris, "Genemania: a real-time multiple association network integration algorithm for predicting gene function," Genome biology, vol. 9, no. 1, p. S4, 2008.

[53] Y. Prabhu and M. Varma, "Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014, pp. 263–272.

[54] H. Jain, Y. Prabhu, and M. Varma, "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016, pp. 935–944.

[55] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma, "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising," in The Web Conference 2018, 2018, pp. 993–1002.

[56] Y. Tagami, "Annexml: Approximate nearest neighbor search for extreme multi-label classification," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017, pp. 455–464.

[57] B. Liu, X. Jin, and C. Li, "Deep learning for extreme multi-label text classification," in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2017, pp. 1155–1158.

[58] J. Nam, J. Kim, E. Menci, I. Gurevych, and M. Zhang, "Large-scale multi-label text classification-revisiting neural networks," in Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2014, pp. 437–452.

[59] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[60] W. Chang, H. Yu, K. Zhong, Y. Yang, and I. S. Dhillon, "A modular deep learning approach for extreme multi-label text classification," CoRR, vol. abs/1905.02331, 2019. [Online]. Available: http://arxiv.org/abs/1905.02331

[61] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang, "Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification," CoRR, vol. abs/2101.03305, 2021. [Online]. Available: https://arxiv.org/abs/2101.03305

[62] Y. You, J. Luo, H. Jin, and J. Yang, "Attentionxml: Attention-based multi-label text classification with relative attention loss," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 619–629.

[63] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 115–124. [Online]. Available: https://doi.org/10.1145/3077136.3080834

[64] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," Mathematics of computation, vol. 19, no. 90, pp. 297–301, 1965.

[65] Y. Li and R. Gayler, "A comparison of vector symbolic architectures," in International Conference on Cognitive Computing. Springer, 2018, pp. 98–107.

[66] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological review, vol. 65, no. 6, p. 386, 1958.

[67] J. Morris, Y. Hao, S. Gupta, R. Ramkumar, J. Yu, M. Imani, B. Aksanli, and T. Rosing, "Multi-label hd classification in 3d flash," in 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), 2020, pp. 10–15.

[68] Z.-H. Zhou, "Ensemble methods for multi-label classification," Morgan & Claypool Publishers, vol. 27, no. 3, pp. 309–319, 2012.

[69] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," in Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2012, pp. 317–332.

[70] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Multi-label classification using ensembles of pruned sets," in Proceedings of the 2008 European conference on machine learning and knowledge discovery in databases. Springer, 2008, pp. 50–65.

[71] A. Elisseeff and J. Weston, "Kernel methods for multi-labelled classification and categorical regression problems," in Proceedings of the 2001 International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., 2001, pp. 179–186.

[72] K. Bhatia, P. Jain, P. Kar, M. Varma, and A. Jain, "Sparse local embeddings for extreme multi-label classification," in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016, pp. 1155–1164.

[73] J. Gao, K. Liu, S. He, and H. Deng, "Large-scale multi-label learning with missing labels," IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 10, pp. 1967–1979, 2017.

[74] V. Y. Shen, N. Saito, M. R. Lyu, M. E. Zurko, B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," Proceedings of the 10th international conference on World Wide Web, pp. 285–295, 2001.

[75] Y. Prabhu and M. Varma, "Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2018, pp. 263–272.

[76] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[77] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Advances in neural information processing systems, 2013, pp. 3111–3119.

[78] Z. S. Harris, "Distributional structure," Word, vol. 10, no. 2-3, pp. 146–162, 1954.

[79] J. Mitchell, M. Lapata, and V. Demberg, "Composition in distributional models of semantics," Cognitive Science, vol. 34, no. 8, pp. 1388–1429, 2010.

[80] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in Proceedings of the conference on empirical methods in natural language processing (EMNLP), 2013, pp. 1631–1642.

[81] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in International Conference on Learning Representations, 2016.

[82] Z. Chen and B. E. Shi, "Appearance-based gaze estimation using dilated-convolutions," CoRR, vol. abs/1903.07296, 2019. [Online]. Available: http://arxiv.org/abs/1903.07296

[83] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Microsoft Research, Tech. Rep. MSR-TR-98-14, 1998.

[84] J. R. Quinlan, C4.5: Programs for machine learning. Elsevier, 1993.

[85] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," Journal of Machine Learning Research, vol. 12, pp. 2411–2414, 2011.

[86] P. Manivasakan and S. Arumugam, "A comparative analysis of classification techniques for genbase dataset," International Journal of Applied Engineering Research, vol. 11, no. 3, pp. 1834–1839, 2016.

[87] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR06), vol. 2. IEEE, 2006, pp. 2169–2178.

[88] A. Elisseeff and J. Weston, "Kernel methods for multi-labelled classification and categorical regression problems," in Advances in neural information processing systems, 2002, pp. 681–687.
[89] I. Ali, M. J. Khan, and M. M. Rahman, "Xt: An xgboost-based extreme multi-label text classification system," in Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 2018, pp. 1355–1358.
[90] M. Chen, Z. Xu, K. Q. Weinberger, and F. Sha, "Bonsai: An efficient framework for learning-based hierarchical document classification," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1567–1576.
[91] X. Zhang, J. Zhao, J. Lian, H. Zhang, and X. Hu, "Parabel: Partitioned label trees for extreme classification with incomplete labels," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6617–6628.
[92] C. G. M. Snoek, M. Worring, J. C. van Gemert, J.-M. Geusebroek, and A. W. M. Smeulders, "The challenge problem for automated detection of 101 semantic concepts in multimedia," in ACM Multimedia, 2006, pp. 421–430.
[93] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multilabel text classification for automated tag suggestion," in ECML/PKDD Discovery Challenge, 2008.
[94] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels," in ECML/PKDD 2008 Workshop on Mining Multidimensional Data, 2008.
[95] D. D. Lewis, Y. Yang, T. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," JMLR, 2004.
[96] A. Zubiaga, "Enhancing navigation on wikipedia with social tags," Preprint, 2009.
[97] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in Proceedings of the 7th ACM conference on Recommender systems. ACM, 2013.
[98] M. Lee, D. Kim, and J. Park, "Light-xml: A memory and computation-efficient approach for extreme multi-label text classification," in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 4120–4129.
[99] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Fastxml: A fast and accurate multi-class classifier for xml data," Machine Learning, vol. 102, no. 2, pp. 243–272, 2016.
[100] S. Goyal, S. Pandey, and H. A. Murthy, "Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning," in Proceedings of the 31st International Conference on Machine Learning (ICML). JMLR Workshop and Conference Proceedings, 2014, pp. 193–201.
[101] Y. Prabhu and M. Varma, "Fast and accurate multilabel classification with deep learning and label relations," in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2018, pp. 259–268.
[102] W. Chen, Z. Song, and T.-Y. Liu, "Pfastrexml: Parallelizing fastxml for extreme multi-label text classification," in Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 2018, pp. 1941–1944.
[103] A. Gupta, R. Gupta, S. Verma, S. Agarwal, and M. Varma, "X-transformer: Transformer with expert-level explanations," arXiv preprint arXiv:2012.07023, 2020.
[104] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International conference on machine learning, 2015, pp. 448–456.
[105] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

RISHIKANTH CHANDRASEKARAN (Graduate Student Member, IEEE) received the M.S. degree in Computer Engineering from Columbia University in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA. He is a member of the System Energy Efficiency Lab, where he works on designing lightweight machine learning algorithms using Hyperdimensional Computing. His research interests primarily focus on breaking the compute barrier for learning tasks in IoT applications.

FATEMEH ASGARINEJAD (Graduate Student Member, IEEE) is a PhD student in the department of Electrical Engineering in UC San Diego and San Diego State University. Her primary research interests lie in the fields of hyperdimensional computing and machine learning.

JUSTIN MORRIS (Member, IEEE) Justin Morris recieved his Ph.D. degree from University of California, San Diego and San Diego State University in 2022. He had the pleaure of being advised by both Dr. Tajana Rosing and Dr. Baris Aksanli. Before starting his Ph.D. degree, Justin was an undergraduate student researcher at the University of California, San Diego. He finished his B.S. in 2018 and took a gap year focusing on research while he applied for his Ph.D. to stay in Dr. Tajana Rosing's research group, System Energy Efficiency Lab. During both his undergraduate and graduate degrees, Justin was activly engaged in teaching. He worked as a Tutor, TA, and also an Instructor at the University of California, San Diego. Justin recently joined California State University, San Marcos as an Assistant Professor of Computer Engineering in Fall 2022.

TAJANA ROSING (Member, IEEE) received the M.S. degree in engineering management con- currently with the Ph.D. degree from Stanford University, Stanford, CA, USA, in 2001 with the Ph.D. topic "Dynamic Management of Power Consumption."

She is a Professor, a Holder of the Fratamico Endowed Chair, and the Director of System Energy Efficiency Laboratory, University of California at San Diego, La Jolla, CA, USA. From 1998 to 2005, she was a full-time Research Scientist with HP Labs, Palo Alto, CA, USA, while also leading research efforts with Stanford University, Stanford, CA, USA. She was a Senior Design Engineer with Altera Corporation, San Jose, CA, USA. She is leading a number of projects, including efforts funded by DARPA/SRC JUMP CRISP program with focus on design of accelerators for analysis of big data, DARPA and NSF funded projects on hyperdimensional computing and SRC funded project on IoT system reliability and maintainability. Her current research interests include energy efficient computing, cyber-physical, and distributed systems.