# Exploring Centrality Measures and Encoding Variants for Graph Classification in Hyperdimensional Computing

Ignacio Sica
*Departamento de Informática*
*Universidad Católica del Uruguay*
Montevideo, Uruguay
mignacio.sica@gmail.com

Gustavo Vazquez
*Departamento de Informática*
*Universidad Católica del Uruguay*
Montevideo, Uruguay
0000-0002-5549-7929

***Abstract*—Graph classification plays a central role in many scientific disciplines. Among existing approaches, classical machine learning methods and graph neural networks have demonstrated robust performance, though often at the cost of substantial computational resources. In recent years, Hyperdimensional Computing (HDC) has emerged as an efficient and noise-resilient alternative, offering a lightweight architecture well suited to resource-constrained environments. However, a key challenge in applying HDC to graph classification lies in generating high-dimensional representations that effectively encode the structural patterns and latent information inherent in graphs.**

**This paper builds upon the GraphHD framework by exploring alternative node centrality metrics. In addition, we introduce GraphHD-Level and GraphHD-Order, two novel variants that incorporate centrality information through distinct encoding strategies. Experiments on benchmark datasets from cheminformatics and bioinformatics (PROTEINS, DD, ENZYMES, NCI1, PTC-FM, and MUTAG) demonstrate that the proposed methods achieve classification performance comparable to standard approaches, while significantly reducing encoding time.**

***Index Terms*—hyperdimensional computing, graph, classification, artificial intelligence**

## I. Introduction

Graphs represent one of the most expressive and flexible data structures for modeling complex relationships among entities. Their applicability extends across a wide range of domains, including bioinformatics, computational chemistry, social networks, transportation systems, and anomaly detection [1]. In this context, graph classification has emerged as a task of growing importance. The goal is to learn predictive representations that effectively capture both local and global structural information, enabling the assignment of labels to entire graphs by leveraging their topology and, when available, their node and edge attributes [2].

Although significant progress has been made in Graph Neural Networks (GNNs) and graph kernel methods, graph classification still faces challenges related to the high computational cost of these learning models, their structural complexity, and their limited performance in scenarios with few training samples. In this context, Hyperdimensional Computing (HDC) emerges as a promising alternative. It relies on distributed representations encoded as high-dimensional vectors, which enable efficient, noise-tolerant, and highly parallelizable information processing. Its neuromorphic architecture is inspired by principles of the human brain [3], such as robustness, holistic representation, and randomness, offering a suitable platform for symbolic reasoning and efficient learning [4]–[7]. Moreover, HDC naturally supports incremental or online learning, enabling continuous knowledge updates as new data become available, without the need to retrain models from scratch. This property is especially valuable in dynamic scenarios where problem conditions frequently change.

An initial attempt to apply HDC to graph classification was proposed through the GraphHD algorithm, which uses a simple encoding scheme based on PageRank centrality to assign hypervectors to nodes and combines edges using using operators defined within the HDC framework [8].

While this strategy showed competitive results in terms of accuracy and efficiency compared to traditional methods, it follows a design that does not incorporate node or edge attributes, does not differentiate between directed and undirected graphs, and relies on a fixed encoding scheme that prioritizes simplicity over semantic flexibility.

The general objective of this study is to design and evaluate expressive and efficient hyperdimensional representations for graph classification. To this end, we build upon the GraphHD approach and propose two main variants. First, we assess the impact of replacing PageRank with alternative centrality measures for node-to-hypervector assignment. Second, we introduce two new encoding strategies: GraphHD-Level, which encodes the actual centrality values using level-hypervectors to preserve quantitative structural information; and GraphHD-Order, a simplified variant that eliminates edge encoding altogether. The proposed methods are evaluated on six widely used benchmark datasets—MUTAG, ENZYMES, PROTEINS, DD, NCI1, and PTC FM—selected to enable consistent comparison with existing approaches in the literature. In addition to

We gratefully thank to Agencia Nacional de Investigación e Innovación (ANII-Uruguay) for funding this work through grant number FCE-1-2023-1-176242

reporting classification performance and encoding efficiency, this work also discusses potential limitations of the proposed strategies when applied to real-world scenarios where semantic information beyond graph structure may be essential.

## II. Background and Related Work

### A. Hyperdimensional Computing Overview

Hyperdimensional Computing is a computational paradigm inspired by fundamental properties of the human brain for storing and processing information. It was originally introduced by Pentti Kanerva as an alternative and efficient way to represent data using distributed vectors of very high dimensionality, known as hypervectors [3], [9]. These hypervectors typically contain thousands or tens of thousands of dimensions, which endow them with unique properties such as robustness to noise, tolerance to partial failures, high associative capacity, and strong potential for parallel computation [10]. This approach is a direct analogy to how concepts are represented in the human brain, where information is not stored in individual neurons, but emerges from the joint contribution of many neurons distributed across distinct cortical regions [9].

From a theoretical perspective, HDC can be viewed as a concrete implementation of the general framework of Vector Symbolic Architectures (VSA). VSA provides a computational model for representing symbols, concepts, and complex structures through high-dimensional vectors combined using specifically defined algebraic operations [11], [12]. In this sense, VSA offers the formal foundation and set of abstract principles for symbolic information representation in vector form, while HDC constitutes a practical realization of these ideas, drawing inspiration from neuromorphic principles.

### B. HDC Framework

In HDC, the basic unit of representation is the hypervector, a very high-dimensional vector typically denoted as $\mathbf{v} \in \mathbb{V}^D$, with $D = 10{,}000$ in most practical applications. Also, a set of fundamental operators is defined over these hypervectors, allowing them to be combined and manipulated in meaningful ways [13]. These basic operators form a closed and flexible algebra that enables the construction of robust and efficient symbolic representations. In particular, they allow for the explicit and reversible representation of objects, as well as the creation of complex hierarchical or nested structures, addressing key challenges such as the binding problem and the superposition catastrophe [14], [15].

Hypervectors can be defined over different domains, with the most common being:

- binary: $\mathbb{V} = \{0, 1\}$
- bipolar: $\mathbb{V} = \{-1, +1\}$
- normalized real: $\mathbb{V} = \mathbb{R}$, with $\|\mathbf{v}\| = 1$

A fundamental property of hypervectors is their quasi-orthogonality: when generated at random, hypervectors are nearly orthogonal to each other. As a result, unrelated items are represented by distinct and easily distinguishable vectors. This property underpins the robustness and representational capacity of hyperdimensional computing, even in the presence of noise or superposition.

Building on this foundation, HDC defines the set of core operators over hypervectors as described below.

*a) Bundling ($\oplus$):* The bundling operation allows for combining multiple hypervectors into a single distributed representation that retains similarity with the individual elements. Given a set of hypervectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{V}^D$, the aggregated vector is expressed as

$$\mathbf{v}_{\text{bundle}} = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \cdots \oplus \mathbf{v}_n. \tag{1}$$

This operator is commutative and associative, but not reversible. It is commonly used to represent sets, average patterns, or to construct class-representative vectors. Depending on the domain, bundling is typically implemented as majority voting (in binary or bipolar spaces) or element-wise addition followed by normalization (in real-valued spaces). The result preserves the distributed character of the representation while capturing aggregate information.

*b) Binding ($\otimes$):* The binding operation associates two hypervectors in such a way that the resulting representation is dissimilar to both inputs, yet encodes the association between them. For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{V}^D$, it is defined as:

$$\mathbf{v}_{\text{bind}} = \mathbf{a} \otimes \mathbf{b}. \tag{2}$$

The operator $\otimes$ is reversible: if one of the operands is known, the other can be recovered by applying the binding operation again. This property is essential for representing ordered pairs, role-filler bindings, or key-value symbolic associations. Binding is typically implemented as component-wise XOR (in binary domains), component-wise multiplication (in bipolar or real-valued spaces), or other domain-preserving operations that ensure dissimilarity and invertibility.

*c) Permutation ($\rho$):* Permutation is a unary operator that reorders the components of a hypervector in a deterministic way. In HDC, it is commonly implemented as a fixed circular shift or a predefined random bijection over indices. This operation enables the encoding of position, order, or direction by generating distinct but systematically related hypervectors. Permutation is reversible: there exists an inverse $\rho^{-1}$ such that $\rho^{-1}(\rho(\mathbf{v})) = \mathbf{v}$, which ensures the consistent representation of structured information.

*d) Similarity:* Similarity between hypervectors is evaluated using a similarity function $\text{sim}(\cdot, \cdot)$, which measures the degree of closeness between two distributed representations. This comparison plays a central role in key operations of HDC such as retrieval, classification, and associative memory access. The choice of similarity metric depends on the domain of the hypervectors: for real-valued vectors, cosine similarity is commonly used; for binary hypervectors, similarity is often measured via normalized Hamming distance or binary correlation.

### C. Classification Using HDC

Hyperdimensional Computing enables classification tasks through an alternative approach to traditional machine learn-

ing, which typically relies on optimizing model parameters. Instead, HDC is grounded in symbolic encoding of information, its storage in an associative memory, and retrieval based on similarity. This paradigm is particularly attractive in scenarios involving computational constraints, noisy data, or incremental update requirements.

The classification process in HDC is structured into three main stages: encoding, training, and prediction [13]. In the encoding stage, each data instance is transformed into a hypervector through a domain-specific encoder designed to capture the relevant information (in our case, to represent a graph as a hypervector, as described in the following subsection).

During training, a class-representative hypervector is constructed by aggregating the encoded hypervectors of all training samples belonging to that class. This class representation $\mathbf{C}_c$ is computed using successive bundling operations

$$\mathbf{C}_c = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \cdots \oplus \mathbf{v}_{N_c} \tag{3}$$

where each $\mathbf{v}_i$ corresponds to a training sample of class $c$, and $N_c$ is the total number of samples in that class.

In the prediction stage, the input instance is encoded using the same procedure, yielding a query hypervector $\mathbf{q}$. This vector is then compared to all stored class-representative hypervectors, and the label of the most similar class is assigned based on the similarity measure.

Fig. 1 schematically illustrates this HDC-based classification pipeline, from the encoding of input samples to final prediction via associative memory lookup. This approach offers several advantages. First, training requires no iterative optimization: a single pass over the data is sufficient. Second, as previously mentioned, distributed representations are robust to noise and partial failures. Third, the system supports natural incremental learning by simply bundling newly encoded samples with the corresponding class-representative hypervectors, thus enabling knowledge updates without full model retraining. Finally, the entire pipeline can be implemented in a simple and efficient manner, making it well suited for embedded or resource-constrained applications [12], [16].

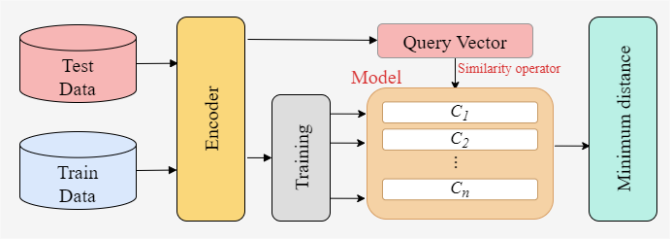


Fig. 1. Schematic overview of the classification process in Hyperdimensional Computing (HDC).

### D. HDC-based Graph Encoding

A graph $G = (V, E)$, with node set $V$ and edge set $E \subseteq V \times V$, is encoded as a hypervector $\mathbf{g} \in \mathbb{V}^D$ through the algebraic composition of hypervectors assigned to its components. This composition is carried out using operations defined within the hyperdimensional computing framework: binding ($\otimes$), bundling ($\oplus$), and permutation ($\rho$).

A representative example for HDC-based graph encoding is GraphHD, which stands as a reference point among early methods [8]. In this approach, each node $v_i \in V$ is assigned a random hypervector $\mathbf{v}_i \in \mathbb{V}^D$, as illustrated in Fig. 2. Each edge $(v_i, v_j) \in E$ is then represented through symmetric binding

$$\mathbf{e}_{ij} = \mathbf{v}_i \otimes \mathbf{v}_j. \tag{4}$$

The graph representation is obtained by aggregating all edge vectors using a bundling operation

$$\mathbf{g} = \bigoplus_{(v_i, v_j) \in E} \mathbf{e}_{ij}. \tag{5}$$

This vector $\mathbf{g}$ serves as a distributed descriptor of the graph and can be directly used for similarity-based classification, as illustrated in Fig. 1.

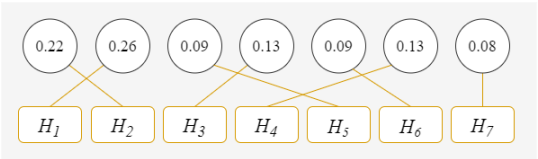


Fig. 2. Assignment of hypervectors in GraphHD.

The GraphHD encoding algorithm includes three main steps for assigning hypervectors to nodes. The first step is the computation of a centrality measure—PageRank ($PR$) in this case. In the second step, vertices are ordered according to the computed values, such that the ordered set $V = \{v_1, v_2, \ldots, v_n\}$ satisfies the following inequality:

$$PR(v_1) \geq PR(v_2) \geq \cdots \geq PR(v_n). \tag{6}$$

In the final step, the ordered vertices are mapped to a list of hypervectors $L = [H_1, H_2, \ldots, H_n]$, which are randomly initialized in advance. This list must have at least the same cardinality as the largest graph in the dataset to ensure a one-to-one assignment. The mapping proceeds as follows:

$$H_1 = v_1, \quad H_2 = v_2, \quad \ldots, \quad H_n = v_n. \tag{7}$$

It is important to note that the centrality values are only used in a relative sense—to establish an ordering within a given graph—while their absolute values are disregarded. Fig. 3 shows a schematic representation of this encoding process. Algorithm 1 presents the general procedure for encoding a graph into a hypervector. When the centrality function $C$ is set to PageRank, the procedure is equivalent to the original GraphHD method. However, $C$ can also be replaced with other centrality measures, as proposed in this work (Section III-A).

Another relevant example is GrapHD [17] that introduces a more flexible encoding scheme that supports representation of node and edge directionality and weights. It also defines

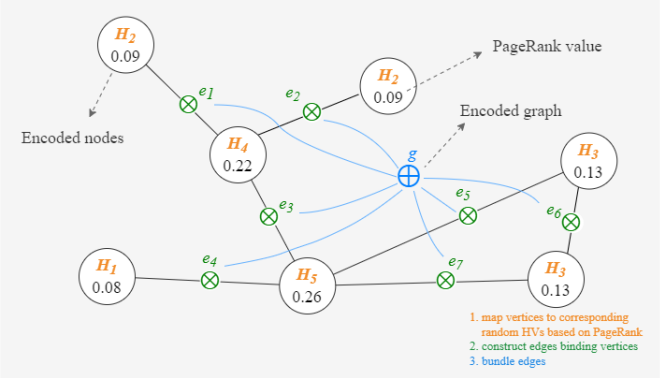


Fig. 3. Illustration of the encoding process in GraphHD. Nodes in the graph are assigned hypervectors according to their PageRank centrality values. Edges are constructed by binding the corresponding node hypervectors, and the entire graph $G$ is represented as the bundling of these encoded edges. The bottom-right corner lists the steps of the encoding process, with actions linked to each step through color coding.

**Algorithm 1** Graph HV Encoding (based on GraphHD)

**Input:** Graph $G = (V, E)$; centrality score function $CS(\cdot)$ (e.g., PageRank, degree, closeness, etc); pre-generated list of hypervectors $\{H_k\}_{k=1}^{M}$ with $M \geq \max_G |V|$.

**Output:** Encoded graph hypervector $g \in V^D$.

1: Compute centrality scores $CS(v)$ for all $v \in V$.

2: Order the vertices $V = \{v_1, \ldots, v_{|V|}\}$ such that
$$\mathrm{CS}(v_1) \geq \mathrm{CS}(v_2) \geq \cdots \geq \mathrm{CS}(v_{|V|}). \quad (6)$$

3: Map the ordered vertices to $L$:
$$H_1 = v_1,\ H_2 = v_2,\ \ldots,\ H_{|V|} = v_{|V|}. \quad (7)$$

4: **for all** $(i, j) \in E$ **do**

5: $e_{ij} \leftarrow V_i \otimes V_j$ ▷ edge encoding (4)

6: $g \leftarrow g \oplus e_{ij}$. ▷ bundle all edges (5)

7: **end for**

8: **return** $g$

cognitive functionalities such as memory reconstruction, information retrieval, graph comparison, and shortest-path search.

In GraphHD, each vertex $v \in V$ is first assigned a hypervector $H_v \in \mathbb{V}^D$. Based on these, a memory hypervector $M_v$ is constructed for each node $v$, capturing its local neighborhood structure. This is defined as the sum (bundle) of the hypervectors of its neighboring nodes $n \in N(v)$:

$$M_v = \sum_{n \in N(v)} H_n \quad (8)$$

where $N(v)$ denotes the set of nodes adjacent to $v$. Fig. 4 illustrates this encoding process.

This aggregation step provides a simple yet effective way to capture the local topology of a graph and serves as the basis for constructing more expressive encodings. Once the memory hypervectors are computed, the final graph representation is obtained by summing the bindings between each node's hypervector and its corresponding memory vector:

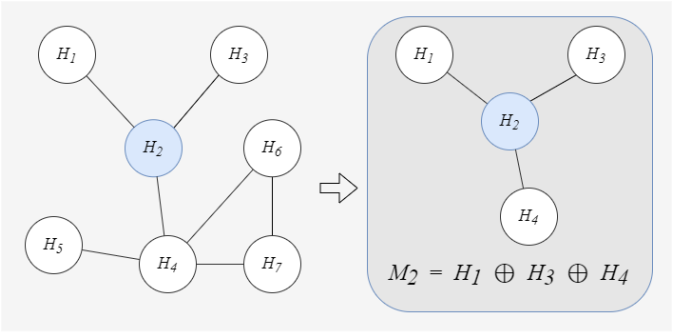


Fig. 4. Example of graph encoding using GrapHD. Hypervectors are assigned to each node ($H_1, H_2$, etc.), and the memory hypervector of node 2 ($M_2$) is built to store connectivity information. The complete encoding of an undirected graph is obtained by aggregating the binding between each node and its memory hypervector.

$$G = \sum_{v \in V} H_v \otimes M_v \quad (9)$$

Although the GrapHD method is not employed in this work, it is described here to provide the reader with an alternative encoding scheme. This offers additional context on possible strategies within the hyperdimensional framework and highlights relevant design choices for graph representation.

It is important to note that all mechanisms discussed in this work encodes graphs primarily at the topological level. When such graphs are abstractions of more complex entities—such as chemical structures—the purely structural representation may not capture all the relevant information.

## III. Methods

Building on GraphHD as a reference method, this work proposes two variants that explore alternative strategies for hyperdimensional graph encoding, focusing on expressiveness and efficiency. The first explores the impact of replacing PageRank with other classic centrality metrics during the encoding process, while the second modifies the way hypervectors are assigned to nodes by replacing the purely order-based scheme with one that leverages the actual centrality values.

### A. Use of Different Centrality Measures

In the original GraphHD approach, the PageRank centrality is used solely to define a relative ranking of nodes, which determines the assignment of pre-generated hypervectors. Thus, the centrality value influences the encoding only indirectly through the order of assignment. In this work, we propose replacing PageRank with other classical centrality measures from graph theory to examine how different structural descriptors influence the resulting representations. To this end, five widely used alternatives are considered, each capturing a distinct aspect of graph structure:

- degree: number of edges incident to a node in an undirected graph, reflecting direct connectivity.

- closeness: average shortest path length from a node to all others, measuring its proximity to the entire graph.
- betweenness: frequency with which a node lies on the shortest paths between other node pairs.
- Katz: weighted count of all paths connecting a node to others, with longer paths receiving less weight.
- eigenvector: centrality based on the importance of a node's neighbors; a node is more central if it connects to other central nodes.

### B. GraphHD-Level: A Value-Based Node Assignment Strategy

GraphHD-Level modifies the original assignment scheme by using the actual values of a centrality measure, rather than their relative order, to guide hypervector assignment. The underlying idea is that the real-valued output of the centrality metric carries relevant structural information, which is ignored when used only for sorting.

This variant follows the general encoding procedure of GraphHD but changes the node-to-hypervector mapping. Instead of assigning hypervectors according to node rank (as in (6)), the goal is to ensure that nodes with similar centrality values are represented by similar hypervectors, as shown in Fig. 5. This is achieved using the level-hypervector method proposed in [18]. The method begins by generating two extreme hypervectors, $L_1$ and $L_m$, and a random vector $\Phi \in \mathbb{R}^D$ with components uniformly distributed in $[0, 1]$. Each intermediate level $L_l$ is constructed via interpolation: the vector $\Phi$ acts as a binary mask that determines for each component whether it should come from $L_1$ or $L_m$, based on a threshold $\tau_l$. Algorithm 2 describes this process.

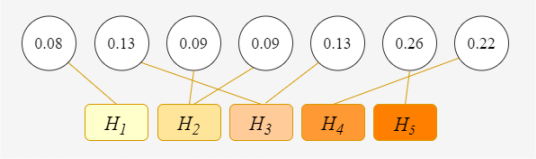


Fig. 5. Node-to-hypervector mapping using level-hypervectors.

**Algorithm 2** Graph HV Encoding (GraphHD-Level variant)

**Input:** Graph $G = (V, E)$; centrality score function $CS(\cdot)$ (e.g., PageRank, degree, closeness, etc); pre-generated set of level-hypervectors $\{L_\ell\}_{\ell=1}^{m}$ covering the possible range of centrality values.

**Output:** Encoded graph hypervector $g \in V^D$.

```
Compute centrality scores CS(v) for all v ∈ V.
for all v ∈ V do
    Determine level index ℓ corresponding to CS(v).
    H_v ← L_ℓ
end for
for all (i, j) ∈ E do
    e_ij ← H_{v_i} ⊗ H_{v_j}
    g ← g ⊕ e_ij
end for
return g
```

The overall graph encoding follows the same procedure as in GraphHD, where the final representation is obtained by bundling the edge hypervectors as defined in (5).

### C. GraphHD-Order: A Simplified Graph Encoding Variant

In GraphHD-Level, the centrality value of each node is encoded directly in its associated hypervector. From this perspective, it may no longer be necessary to explicitly encode graph edges, as node centrality values already reflects structural roles within the graph. This insight motivates the GraphHD-Order variant.

GraphHD-Order skips edge binding and represents the graph as a bundling of node hypervectors—each of which encodes the node's centrality value. This approach improves computational efficiency by eliminating the cost of the neighbour binding operation (4) for edge representations. Naturally, it is only meaningful when hypervectors are assigned using a centrality-preserving strategy such as level-hypervectors (rather than random assignment, as in the original GraphHD). Fig. 6 illustrates the simplified encoding process.

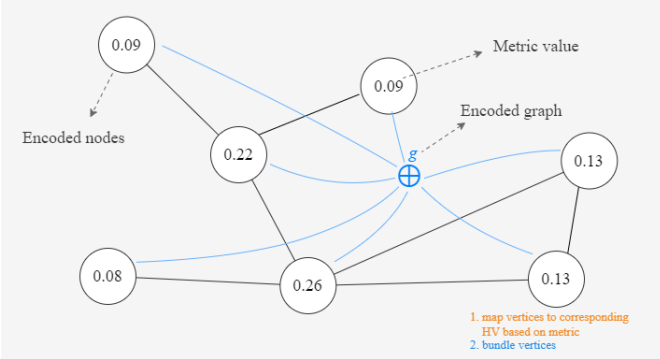


Fig. 6. GraphHD-Order: nodes, shown with their centrality values, are mapped to corresponding level-hypervectors, which are then bundled to produce the final graph representation $G$.

This variant serves as a testbed to explore the expressive power of hyperdimensional computing, investigating how much the encoding process can be simplified without significantly degrading classification performance. Algorithm 3 describes this process.

### D. Datasets

The experimental analysis was conducted using the same benchmark graphs as in the original GraphHD study [8], enabling fair and consistent comparisons. All data are publicly available through the TUDataset repository [19].

The selected benchmarks include PROTEINS and DD (binary classification of protein structures), ENZYMES (assignment of proteins to one of six enzyme commission classes), NCI1 (classification of chemical compounds based on anticancer activity), PTC_FM (prediction of carcinogenicity in rats), and MUTAG (classification of mutagenic aromatic compounds). Table I summarizes key structural properties. It is important to note that only graph connectivity is considered in the encoding process. This work focuses exclusively on graph

**Algorithm 3** GraphHD-Order HV Encoding

**Input:** Graph $G = (V, E)$; centrality score function $CS(\cdot)$ (e.g., PageRank, degree, closeness, etc); set of level-hypervectors $\{L_\ell\}_{\ell=1}^{m}$ covering the possible range of centrality values.

**Output:** Encoded graph hypervector $g \in V^D$.

Compute centrality scores $CS(v)$ for all $v \in V$.
**for all** $v \in V$ **do**
    Determine level index $\ell$ corresponding to $CS(v)$.
    $H_v \leftarrow L_\ell$
**end for**
**for all** $v \in V$ **do**
    $g \leftarrow g \oplus H_v$
**end for**
**return** $g$

topological information, without incorporating node or edge attributes, in order to maintain compatibility with the baseline method and ensure fair comparisons.

TABLE I
DATASET STATISTICS

| Dataset | Graphs | Classes | Avg. Vertices | Avg. Edges |
|---|---|---|---|---|
| DD | 1178 | 2 | 284.32 | 715.66 |
| ENZYMES | 600 | 6 | 32.63 | 62.14 |
| MUTAG | 188 | 2 | 17.93 | 19.79 |
| NCI1 | 4110 | 2 | 29.87 | 32.30 |
| PROTEINS | 1113 | 2 | 39.06 | 72.82 |
| PTC_FM | 349 | 2 | 14.11 | 14.48 |

## IV. Results and Discussion

This section presents the experimental results of the proposed encoding variants and examines the impact of using different centrality measures on classification performance. Fig. 7 reports the F1-score obtained by the original GraphHD method and the two proposed variants, GraphHD-Level and GraphHD-Order, across all datasets. Overall, the results indicate that all methods achieve comparable classification performance.

Since the experimental setup replicates the datasets from the original GraphHD study, the results remain comparable. This consistency ensures that the proposed variants—GraphHD-Level and GraphHD-Order—can be meaningfully contrasted not only with the original GraphHD method, but also with the broader set of models evaluated in that prior work [8]. These include classical kernel-based approaches (1-WL and WL-OA) [20]) and graph neural networks (such as GIN and GIN-JK) [21]). Previous comparisons demonstrated that GraphHD offers competitive classification performance while requiring significantly lower training times. In this context, our results indicate that the proposed variants achieve predictive performance on par with established machine learning methods for graph classification, further validating their effectiveness within the broader landscape of graph-based learning. Notably, GraphHD-Order achieves this performance using a simplified encoding scheme, highlighting the potential of alternative

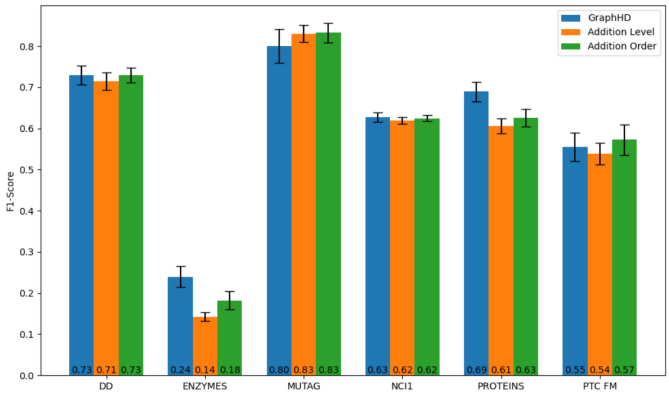


Fig. 7. F1-score for the original GraphHD method and the proposed variants (GraphHD-Level and GraphHD-Order) across different datasets. Results are averaged over 100 independent runs.

TABLE II
COMPARISON OF GRAPHHD AND GRAPHHD-LEVEL USING DIFFERENT CENTRALITY METRICS

| Metric | GraphHD | | | GraphHD-Level | | |
|---|---|---|---|---|---|---|
| | *Mean* | *Std* | *Time* | *Mean* | *Std* | *Time* |
| Pagerank | 0.80 | 0.022 | 1.5e-3 | 0.85 | 0.029 | 2.3e-3 |
| Degree | 0.79 | 0.029 | 4.0e-4 | 0.28 | 0.091 | 4.6e-4 |
| Closeness | 0.80 | 0.028 | 6.5e-4 | 0.77 | 0.025 | 1.1e-3 |
| Betweenness | 0.83 | 0.040 | 6.6e-4 | 0.82 | 0.022 | 8.0e-4 |
| Katz | 0.81 | 0.031 | 6.0e-4 | 0.83 | 0.028 | 6.2e-4 |
| Eigenvector | 0.82 | 0.047 | 9.0e-4 | 0.85 | 0.033 | 9.0e-4 |

centrality-driven representations to reduce computational cost without compromising accuracy.

We now focus on a detailed comparison of classification performance across different centrality measures, using the MUTAG dataset as a representative case. Table II shows the $F_1$ scores and encoding times for both GraphHD and GraphHD-Level using six centrality metrics: PageRank, degree, closeness, betweenness, Katz, and eigenvector. Since centrality metrics encode topological characteristics of graphs, the impact on performance is expected to vary depending on the structural and chemical features of the dataset. Experimental results show that classification performance remains relatively stable across most metrics, and that GraphHD-Level achieves comparable results to the original method. Notably, the proposed centrality variants demonstrate faster encoding times, especially when replacing the PageRank metric. A plausible explanation for the poor performance of degree centrality in GraphHD-Level is that degree is a purely local measure, which often yields low variability among nodes in the same graph. In GraphHD-Level, where centrality values directly determine the assigned level-hypervectors, such low variability leads to highly similar node representations. This reduces the discriminative power of the resulting graph hypervector, particularly in datasets where global or higher-order structural patterns are critical for classification. All experiments reported in Table II were conducted on a standard desktop computer with an Intel Core i7 processor.

Although the proposed methods achieve comparable clas-

sification performance and improved encoding times, it is important to note a key limitation: all approaches -including GraphHD- are based solely on the structural information of graphs, without incorporating any chemical or biological features. This limitation is particularly evident in the poor performance observed for the ENZYMES dataset (Fig. 7). None of the encoding schemes are capable of capturing functionally relevant motifs or active-site patterns required for enzyme classification, where each molecule belongs to one of six major groups defined by the type of chemical reaction it catalyzes. As a result, two molecules with distinct enzymatic activity may exhibit highly similar topological structures and thus be mapped to nearly identical hypervectors, making it difficult to distinguish them without incorporating additional chemical or functional information.

Interestingly, the opposite is true for the MUTAG dataset. As seen in Table II, eigenvector centrality performs best among all metrics. This is likely due to the chemical nature of MUTAG, where both local and structural features are crucial in determining mutagenic activity. Eigenvector centrality is particularly effective at capturing stable substructures and influential node configurations [22], [23].

The implementation of core scripts are available at: https://github.com/gustavovazquez/HDC-Graph-Classification.

## V. Conclusion and Future Work

This work investigated the application of Hyperdimensional Computing (HDC) to graph classification, introducing two novel encoding variants—GraphHD-Level and GraphHD-Order—that simplify and extend the original GraphHD method. In addition, the study explored the impact of various centrality measures on graph representation.

Experimental results demonstrate that both proposed variants achieve competitive classification performance while reducing encoding times, particularly in the case of GraphHD-Order, which omits edge encoding entirely. The analysis also reveals that the choice of centrality metric influences classification outcomes.

These findings support the potential of HDC-based models as lightweight and efficient alternatives to more complex kernel-based or neural approaches. However, the results also underscore a key limitation: in many cases, the topological structure of a graph alone is not sufficient to capture the functional or semantic properties required for accurate classification. This is particularly evident in datasets where domain-specific information—such as chemical, biological, or functional attributes—plays a critical role.

Several directions for future research emerge from this study. One is to enrich the graph encoding with domain-specific attributes, allowing the representation to capture more nuanced information beyond topological structure. Another involves integrating attention mechanisms into the HDC framework to enable selective weighting of graph components during encoding. Finally, the symbolic reasoning capabilities of HDC offer opportunities for addressing tasks such as similarity search, motif detection, and explainability analysis in scientific domains.

## VI. Acknowledgement

As suggested by IEEE Author Center (Submission and Peer Review Policies) the authors disclose the use of ChatGPT (OpenAI) for grammar and style refinement. All scientific content remains the sole work of the authors.

## References

[1] A. Bondy and U. S. R. Murty, *Graph Theory*. New York: Springer, 2008.

[2] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph Learning: A Survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, Apr. 2021. [Online]. Available: https://ieeexplore-ieee-org.proxy.timbo.org.uy/document/9416834

[3] P. Kanerva, "Fully Distributed Representation," *Real World Computing Symposium (RWC)*, pp. 358–365, 1997. [Online]. Available: http://www.cap-lore.com/RWC97-kanerva.pdf

[4] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges," *ACM Computing Surveys*, vol. 55, no. 9, pp. 175:1–175:52, Jan. 2023. [Online]. Available: https://doi.org/10.1145/3558000

[5] M. Heddes, I. Nunes, T. Givargis, A. Nicolau, and A. Veidenbaum, "Hyperdimensional computing: a framework for stochastic computation and symbolic AI," *Journal of Big Data*, vol. 11, no. 1, p. 145, Oct. 2024. [Online]. Available: https://doi.org/10.1186/s40537-024-01010-8

[6] F. Cumbo and D. Chicco, "Hyperdimensional computing in biomedical sciences: a brief review," *PeerJ Computer Science*, vol. 11, p. e2885, May 2025, publisher: PeerJ Inc. [Online]. Available: https://peerj.com/articles/cs-2885

[7] J. E. Q. Ibarra, J. Á. G. Ordiano, J. J. Flores Godoy, and G. E. Vazquez, "Clustering of News Texts Using Hyperdimensional Computing," in *2024 IEEE URUCON*, Nov. 2024, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10850161

[8] I. Nunes, M. Heddes, T. Givargis, A. Nicolau, and A. Veidenbaum, "GraphHD: efficient graph classification using hyperdimensional computing," in *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, ser. DATE '22. Leuven, BEL: European Design and Automation Association, May 2022, pp. 1485–1490.

[9] 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, Jun. 2009. [Online]. Available: https://doi.org/10.1007/s12559-009-9009-8

[10] T. Plate, *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Publications, 2003. [Online]. Available: https://web.stanford.edu/group/cslipublications/cslipublications/site/1575864304.shtml

[11] S. D. Levy and R. Gayler, "Vector Symbolic Architectures: A New Building Material for Artificial General Intelligence," in *Proceedings of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*. NLD: IOS Press, Jun. 2008, pp. 414–418.

[12] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, and F. T. Sommer, "Vector Symbolic Architectures as computing framework for nanoscale hardware," *Proceedings of the IEEE*, Oct. 2022, publisher: IEEE.

[13] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 130:1–130:40, Dec. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3538531

[14] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," in *Psychology of Learning and Motivation*, G. H. Bower, Ed. Academic Press, Jan. 1989, vol. 24, pp. 109–165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079742108605368

[15] T. Plate, "Holographic reduced representations," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–641, May 1995, conference Name: IEEE Transactions on Neural Networks.

[16] Y. Huang, A. J. Rad, and Q. Xia, "Hardware-Algorithm Co-Design for Hyperdimensional Computing Based on Memristive System-on-Chip," in *Proceedings NeurIPS2024*, Oct. 2024. [Online]. Available: https://openreview.net/forum?id=rRIZblLJHb

[17] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, and M. Imani, "GrapHD: Graph-Based Hyperdimensional Memorization for Brain-Like Cognitive Learning," *Frontiers in Neuroscience*, vol. 16, Feb. 2022, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.757125/full

[18] I. Nunes, M. Heddes, T. Givargis, and A. Nicolau, "An Extension to Basis-Hypervectors for Learning from Circular Data in Hyperdimensional Computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10247736

[19] "TUDataset." [Online]. Available: https://chrsmrrs.github.io/datasets/datasets/

[20] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, Dec. 2020, number: 1 Publisher: SpringerOpen. [Online]. Available: https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0195-3

[21] L. Wei, H. Zhao, Z. He, and Q. Yao, "Neural Architecture Search for GNN-Based Graph Classification," *ACM Trans. Inf. Syst.*, vol. 42, no. 1, pp. 1:1–1:29, Aug. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3584945

[22] N. Parisutham and N. Rethnasamy, "Eigenvector centrality based algorithm for finding a maximal common connected vertex induced molecular substructure of two chemical graphs," *Journal of Molecular Structure*, vol. 1244, p. 130980, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022286021011121

[23] C. F. A. Negre, U. N. Morzan, H. P. Hendrickson, R. Pal, G. P. Lisi, J. P. Loria, I. Rivalta, J. Ho, and V. S. Batista, "Eigenvector centrality for characterization of protein allosteric pathways," *Proceedings of the National Academy of Sciences*, vol. 115, no. 52, pp. E12 201–E12 208, Dec. 2018, publisher: Proceedings of the National Academy of Sciences. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.1810452115