Beyond PageRank in GraphHD: Centrality Metrics and Efficient Hyperdimensional Encodings

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

and

Gustavo Vazquez *
Departamento de Informática e Inteligencia Artificial,
Universidad Católica del Uruguay,
Montevideo, Uruguay,
gustavo.vazquez@ucu.edu.uy

Abstract

Graph classification plays a central role in many scientific disciplines. While classical kernel-based methods and graph neural networks achieve strong predictive performance, they often require substantial computational resources. Hyperdimensional Computing (HDC) has recently emerged as an efficient and noise-resilient alternative, providing lightweight models that are attractive for resource-constrained settings. Within this context, GraphHD is a representative HDC-based approach for graph classification; however, its encoding process can become costly on large graphs and its standard configuration relies on a single centrality choice (PageRank) for node-to-hypervector assignment.
In this work, we go beyond PageRank in GraphHD by systematically evaluating alternative centrality measures (degree, closeness, betweenness, Katz, and eigenvector) and by introducing two new encoding variants. GraphHD-Level preserves quantitative structural information by mapping centrality values to level-hypervectors, whereas GraphHD-Order simplifies the algorithm by eliminating edge encoding and aggregating node hypervectors directly. Experiments on six widely used benchmarks from cheminformatics and bioinformatics (MUTAG, ENZYMES, PROTEINS, DD, NCI1, and PTC_FM) show that replacing PageRank with alternative centralities yields similar F1-scores while offering notable runtime savings, and that GraphHD-Order remains competitive with the original GraphHD baseline while providing consistent speedups in encoding time.
*Corresponding author: gustavo.vazquez@ucu.edu.uy

Keywords: hyperdimensional computing, graph, classification, artificial intelligence.

1 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, 5, 6, 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].

A related line of work is Configurable Hyperdimensional Graph Representation [9], which introduces a more flexible and parameterizable graph encoding framework within HDC. In contrast to GraphHD, its design is oriented toward retaining sufficient information to enable (partial) graph reconstruction from the hyperdimensional representation. This emphasis on reconstructability requires a richer encoding procedure, which increases methodological complexity and typically entails a higher computational cost during encoding. Given our focus on lightweight encodings for graph classification, rather than on reconstruction-oriented representations, we do not adopt this configurable reconstruction-driven approach in our experiments, as its additional complexity and encoding cost are not justified for the objectives of this work.

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 extensions. 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.

This journal article extends our previous conference version [10], where the core ideas were first introduced. Here, we provide a more comprehensive experimental study by combining each centrality measure with both proposed encoding variants, and we report additional classification and runtime results. We also expand the discussion to better characterize the behavior of the methods and the trade-offs between accuracy and computational efficiency. Finally, we release an updated implementation of all methods, including command-line parameterization, CPU-level parallelization, and additional code improvements.

2 Background and Related Work

2.1 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[11, 3]. 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 [12]. 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[11].

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[13, 14]. 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.

2.2 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[15]. 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 [16, 17].

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.

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.

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.

Permutation (ρ) 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 ρ^{-1} such that $\rho^{-1}(\rho(\mathbf{v})) = \mathbf{v}$, which ensures the consistent representation of structured information.

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.

2.3 Classification Using HDC

Hyperdimensional Computing enables classification tasks through an alternative approach to traditional machine learning, 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 [15]. 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[18, 14].

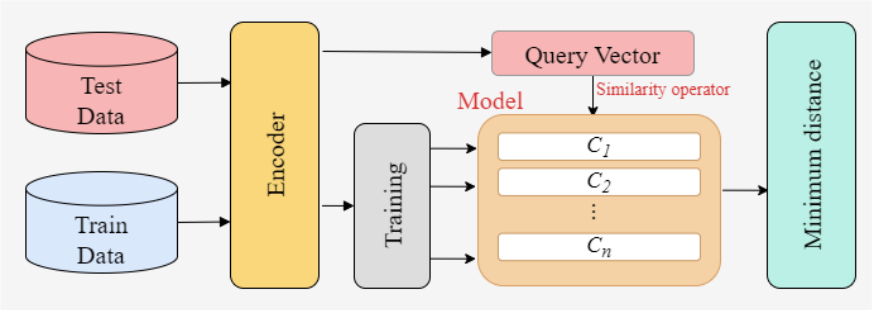


Figure 1: Schematic overview of the classification process in Hyperdimensional Computing (HDC)

2.4 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 (ρ).

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.

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:

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

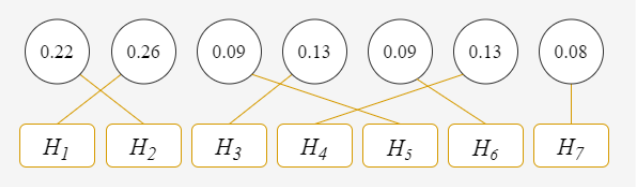


Figure 2: Assignment of hypervectors in GraphHD

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 3.1).

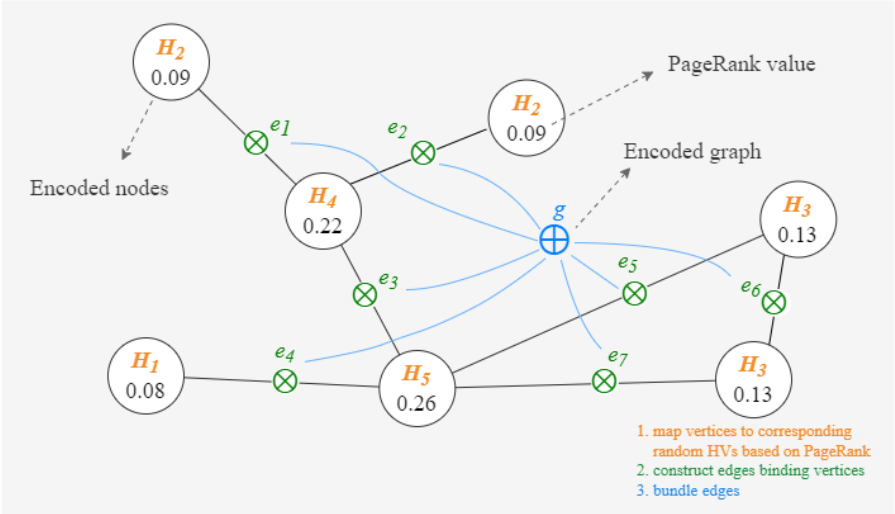


Figure 3: Illustration of the encoding process in GraphHD. Nodes in the graph are assigned hypervectors according to their PageRank centrality values. Edges are constructed by binding the corresponding node hypervectors, and the entire graph G is represented as the bundling of these encoded edges. The 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)

Require: 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 $M \geq \max_G |V|$.
Ensure: 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
$\mathrm{CS}(v_1) \geq \mathrm{CS}(v_2) \geq \cdots \geq \mathrm{CS}(v_{|V|})$. (6)
3: Map the ordered vertices to L:
$H_1 = v_1,\ H_2 = v_2,\ \ldots,\ H_{|V|} = v_{|V|}$. (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

Another relevant example is GrapHD [19] that introduces a more flexible encoding scheme that supports representation of node and edge directionality and weights. It also defines cognitive functionalities such as memory reconstruction, information retrieval, graph comparison, and shortest-path search.

In GrapHD, 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 $n \in N(v)$:

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

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

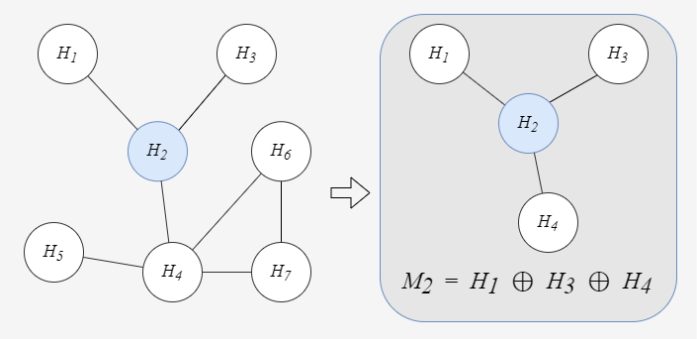


Figure 4: Example of graph encoding using GrapHD. Hypervectors are assigned to each node (H_1, H_2, etc.), and the memory hypervector of node 2 (M_2) is built to store connectivity information. The complete encoding of an undirected graph is obtained by aggregating the binding between each node and its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

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:

$$G = \sum_{v \in V} H_v \otimes M_v \tag{9}$$

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.

3 Methods

Building upon GraphHD as a reference method, this paper extends the original encoding scheme by incorporating different graph centrality measures and introduces two novel hyperdimensional graph encoding algorithms: GraphHD-Level and GraphHD-Order. The use of alternative centrality metrics is motivated by the fact that those measures capture different structural properties of graphs, which may lead to more expressive representations. Importantly, these centrality measures are applicable both to the original GraphHD method and to the proposed variants, enabling a unified framework for evaluating their impact across different encoding strategies.

3.1 Use of New 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.

Collectively, these measures provide complementary structural information: degree emphasizes local connectivity, closeness captures global accessibility, betweenness highlights brokerage roles between regions, while Katz and eigenvector incorporate indirect influence through multi-hop connectivity, the latter weighting connections to already central neighbors. Leveraging them within the same encoding pipeline enables the learned representation to probe different graph characteristics without modifying the underlying hypervector coding method. Consequently, the choice of centrality measure can be formalized as a model hyperparameter, to be selected (or tuned) depending on the task and dataset.

3.2 GraphHD-Level: A New 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 [20]. 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 Φ acts as a binary mask that determines for each component whether it should come from L_1 or L_m, based on a threshold τ_l. Algorithm 2 describes this process.

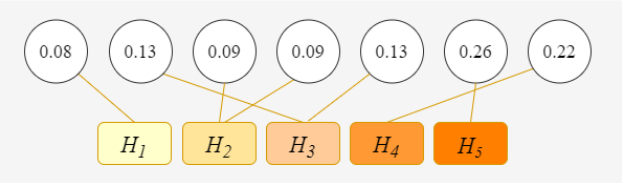


Figure 5: Node-to-hypervector mapping using level-hypervectors

Algorithm 2 Graph HV Encoding (GraphHD-Level variant)

Require: 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.
Ensure: 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). Importantly, the encoding process is agnostic

to the specific centrality measure employed, allowing any of the centrality metrics introduced in Section 3.1 to be seamlessly integrated without altering the underlying encoding pipeline.

3.3 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, where the centrality metric can be chosen from those presented in Section 3.1. This approach improves computational efficiency by eliminating the cost of the neighbor binding operation in (4) used for edge representations. Fig. 6 illustrates the simplified encoding process.

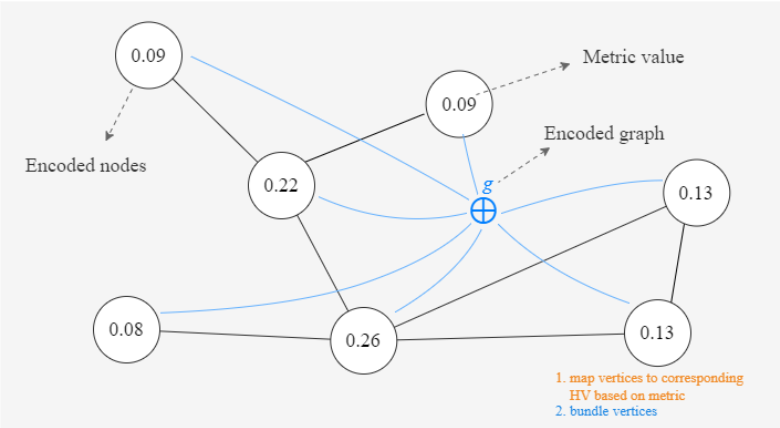


Figure 6: GraphHD-Order: nodes, shown with their centrality values, are mapped to corresponding level-hypervectors, which are then bundled to produce the final graph representation G

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.

Algorithm 3 GraphHD-Order HV Encoding

Require: 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.
Ensure: 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 v ∈ V do
    g ← g ⊕ H_v
end for
return g
```

3.4 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 [21].

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 anti-cancer activity), PTC_FM (prediction of carcinogenicity in rats), and MUTAG (classification of mutagenic aromatic compounds). Table 1 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 topological information, without incorporating node or edge attributes, in order to maintain compatibility with the baseline method and ensure fair comparisons.

Table 1: 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

3.5 Implementation Details and Experimental Setup

Since hypervector initialization is inherently random, all experiments were run with 100 independent repetitions, and we report the mean and standard deviation of each metric across them. Parallelism is exploited at the level of the experiment grid: each *(encoder, centrality measure)* configuration—together with all its repetitions—is dispatched to a separate worker process through a process pool, so that independent configurations are executed concurrently across CPU cores. The encoding of an individual graph is itself carried out sequentially using vectorized NumPy operations. Crucially, the same parallelization scheme is applied identically to every encoder variant, so the runtimes reported in Section 4 are directly comparable across methods. The experiments were executed on a desktop PC equipped with an AMD Ryzen 9 5950X CPU (16 cores; 32 threads were used for the runs) and 64GB of RAM.

With respect to the implementation, all scripts were developed in Python. In all methods hypervectors were encoded using a bipolar representation $\{-1, +1\}$ (Section 2.2). For efficiency, hypervectors were stored as NumPy arrays with dtype `int8`, which reduces the memory footprint. The binding operation was implemented as element-wise multiplication, and bundling was performed by summing the operand hypervectors and then applying the $sign()$ function to project the result back to the bipolar space. The hypervector dimensionality was set to 10,000, following common practice in hyperdimensional computing. The implementation of scripts, datasets and detailed execution results are available as supplementary material at https://github.com/gustavovazquez/CLEI-2025.

Regarding the validation protocol, for every dataset, each repetition draws an independent random partition of the graphs into 80% training and 20% test sets (hold-out evaluation). The encoded training graphs are bundled into one prototype hypervector per class, and each test graph is assigned the label of the most similar class prototype under cosine similarity, as indicated in Fig. 1. As performance metrics, we report the weighted F1-score and accuracy. No separate validation set is required since HDC is hyperparameters free. Because the differences observed across centrality measures and encoder variants are small, we further assess their statistical significance with a *paired* protocol: within each repetition a single train/test split is shared by all methods, so their F1-scores are directly comparable, and we contrast pairs of methods with the Wilcoxon signed-rank test, applying the Holm–Bonferroni correction for multiple comparisons. The corresponding analysis is reported in next Section.

4 Results and Discussion

This section reports the experimental results of the proposed encoding variants and analyzes the effect of different centrality measures on classification performance. To organize the comparisons, we first examine the impact of the centrality measure originally used in GraphHD and compare it with the alternative measures introduced in Section 3.1. We then extend this analysis by presenting results for the two proposed variants, GraphHD-Order and GraphHD-Level, and discussing their behavior under the same set of centrality choices. Section 4.3 reports the statistical significance analysis.

4.1 Effect of Different Centrality Measures

Since our experimental setup replicates the datasets used in the original GraphHD study [8], the results reported here remain directly comparable to those previously published. In that original work, GraphHD demonstrated excellent performance against classical kernel-based approaches (e.g., 1-WL and WL-OA)[22] and graph neural networks (e.g., GIN and GIN-JK)[23]. These comparisons showed that GraphHD achieves

competitive classification accuracy while requiring substantially lower training times than state-of-the-art methods. Therefore, we adopt GraphHD as the primary reference point throughout our experiments.

So, in what follows, we compare GraphHD's original PageRank-based centrality measure with the alternative measures introduced in Section 3.1 (degree, closeness, betweenness, Katz, and eigenvector centrality). The resulting F1-scores are reported in Table 2, where the PageRank column corresponds to the original GraphHD configuration and is used as a reference. Overall, the results show that replacing PageRank with alternative centrality measures has a limited impact on GraphHD classification performance across the evaluated datasets. In most cases, the F1-scores obtained with degree, closeness, betweenness, Katz, or eigenvector centrality remain very close to those achieved with the PageRank-based baseline, differing on average by only about 0.010 in absolute value (at most 0.034), with no single centrality proving consistently best. The corresponding paired significance analysis is reported in Section 4.3.

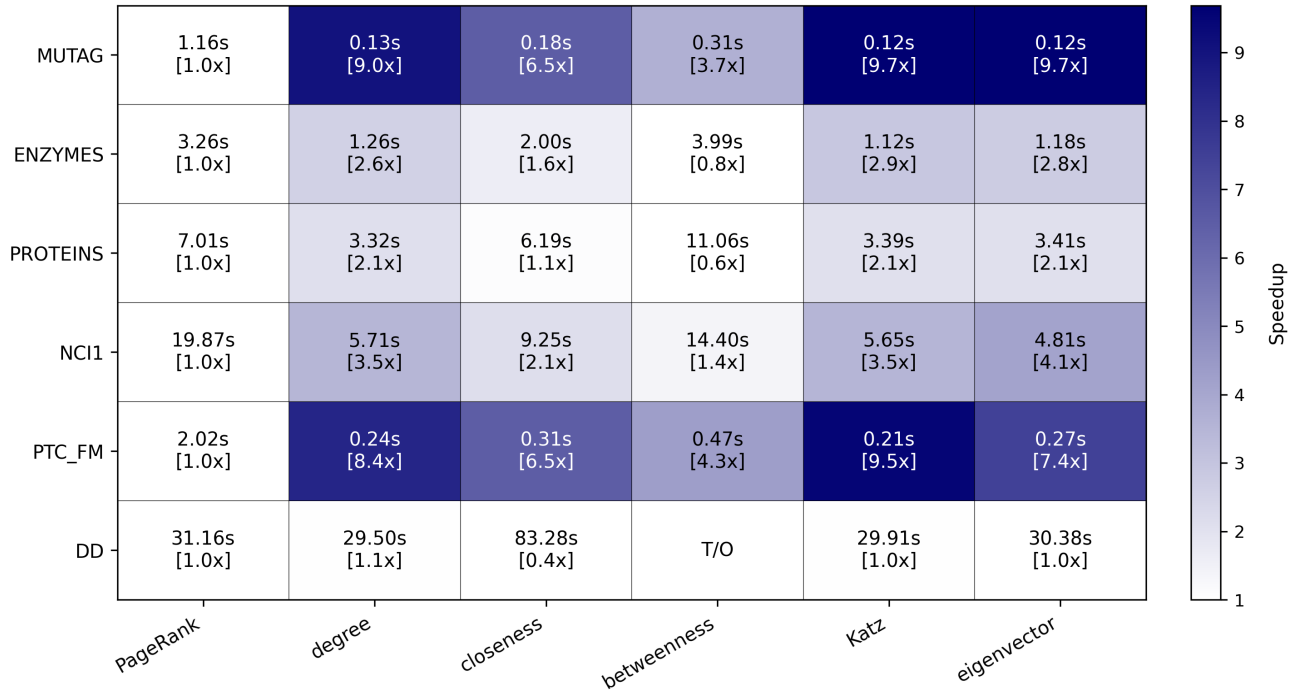
Although PageRank can be seen as a richer global descriptor at the node level, this advantage does not necessarily translate into noticeably better graph-level accuracy. Even simpler measures such as degree may be less informative for individual nodes, yet the final graph representation is obtained by bundling the hypervectors of all nodes and edges, effectively aggregating connectivity information at scale. As a result, different notions of node importance -local or global- tend to yield graph encodings with comparable discriminative power once integrated through the same HDC algorithm. This behavior is expected, since all these centrality measures capture, in different ways, topological information about node importance within the graph (e.g., local connectivity, proximity, brokerage roles, or multi-hop influence).

We next analyze computational performance in terms of execution time. While the F1-scores reported in Table 2 are insensitive to the choice of centrality measure, the runtimes reported in Fig. 7 (time/speedup heatmap) reveal substantial differences in computational cost (additional info (e.g., std. dev.) is included in the supplementary material). In particular, degree, Katz, and eigenvector centrality consistently yield the fastest executions across datasets, often providing multi-fold speedups over the original PageRank configuration (as indicated by $[T_{\text{PageRank}}/T_{\text{metric}}]$). In contrast, closeness and especially betweenness incur a markedly higher computational time, which becomes dominant on larger datasets, reflecting their reliance on shortest-path computations. Overall, these results suggest that, since accuracy remains comparable, the choice of centrality measure can be guided primarily by efficiency considerations. In particular, our proposal of replacing GraphHD's original PageRank with alternative centrality measures yields a clear practical benefit: in our implementation, degree, Katz, and eigenvector centrality provide the most attractive accuracy-runtime trade-off.

Table 2: F1 mean for GraphHD using different centrality metrics

Dataset	PageRank	degree	closeness	betweenness	Katz	eigenvector
DD	0.685	0.685	0.687	T/O[a]	0.685	0.685
ENZYMES	0.277	0.284	0.300	0.267	0.279	0.277
MUTAG	0.873	0.857	0.842	0.857	0.857	0.857
NCI1	0.636	0.624	0.627	0.618	0.627	0.627
PROTEINS	0.733	0.721	0.732	0.722	0.721	0.721
PTC_FM	0.572	0.572	0.572	0.580	0.574	0.571

[a] T/O: exceeded the time limit (runtime too high).



T/O: exceeded time limit (runtime too high).

Figure 7: GraphHD mean execution time(s) and speedup $[T_{\text{PageRank}}/T_{\text{metric}}]$ relative to PageRank across centrality measures and datasets.

4.2 New Methods: GraphHD-Level and GraphHD-Order

We now extend the previous analysis to the two proposed variants, GraphHD-Level and GraphHD-Order, and evaluate their classification performance under the same set of centrality measures considered. While results are available for all metrics, reporting the full set of plots would be unnecessarily redundant; therefore, we focus on a representative subset of three measures that capture complementary. Specifically, we include PageRank as the original GraphHD choice and a natural reference point, degree as a simple and computationally inexpensive local measure, and betweenness as a more global shortest-path–based measure that typically represents the most demanding computational case. The remaining centralities (closeness, Katz, and eigenvector) are included in the GitHub repository.

We begin with the PageRank setting (Fig. 8), since corresponds to the original configuration and thus provides a natural baseline. Under this reference setup, GraphHD-Order consistently matches or improves upon the baseline across datasets, and it delivers the clearest gains on the largest dataset DD (this phenomenon will be discussed in next subsection). In contrast, GraphHD-Level does not exhibit the same permormance: its F1-scores are generally below those of GraphHD and GraphHD-Order, suggesting that the proposed level-based assignment does not translate into consistent improvements in classification performance. Overall, these results highlight GraphHD-Order as the variant that remains competitive with the original GraphHD baseline.

Notably, the same behavior is observed when replacing PageRank with the other centrality metrics: GraphHD-Order remains the variant that most consistently matches with the GraphHD baseline, whereas GraphHD-Level typically lags behind and does not yield systematic gains. Figures 9 and 10 show the corresponding results for degree and betweenness centrality, respectively. The complete set of results for the remaining centrality measures is available in the GitHub repository.

We next analyze computational performance in terms of execution time, focusing on the comparison between GraphHD and GraphHD-Order, which emerge as the most competitive methods in terms of F1-score. As expected, the runtime results (reported as time/speedup heatmaps) reveal substantial computational advantages in favor of GraphHD-Order, with speedups typically ranging from approximately 1.1× up to about 3× depending on the dataset. Again, we report execution times for the selected centrality measures in Fig. 11 for PageRank (a), degree (b), and betweenness (c), respectively. The complete set of results for the remaining centrality measures is available in the GitHub repository.

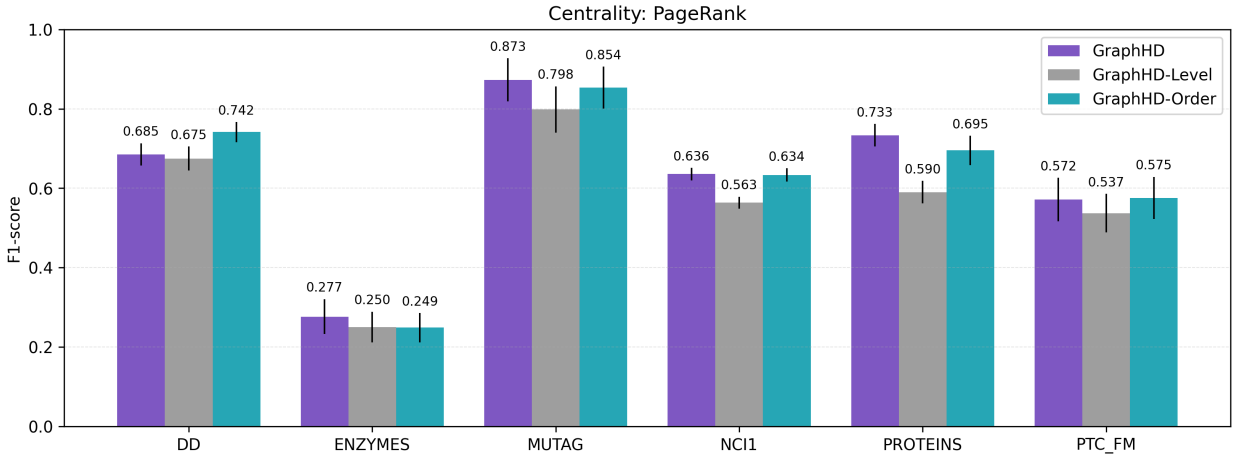


Figure 8: Mean F1-score across datasets using PageRank-based centrality metric. Results are shown for GraphHD, GraphHD-Level, and GraphHD-Order

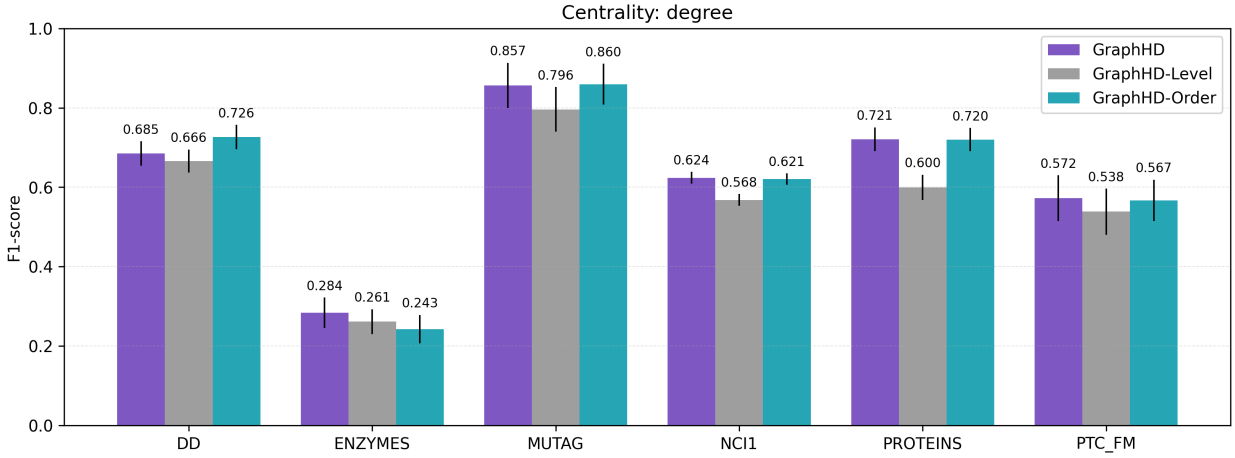


Figure 9: Mean F1-score across datasets using degree-based centrality metric. Results are shown for GraphHD, GraphHD-Level, and GraphHD-Order

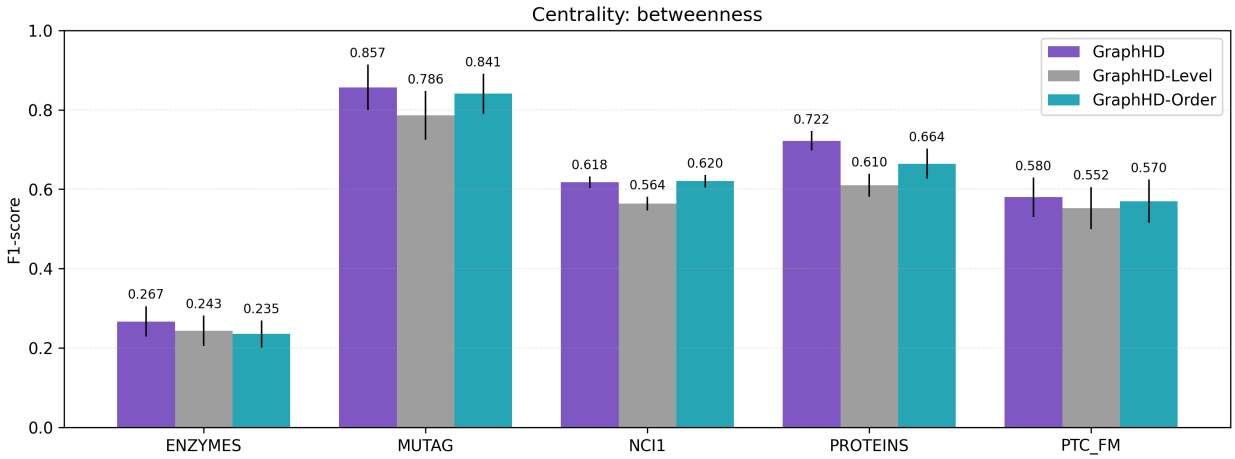
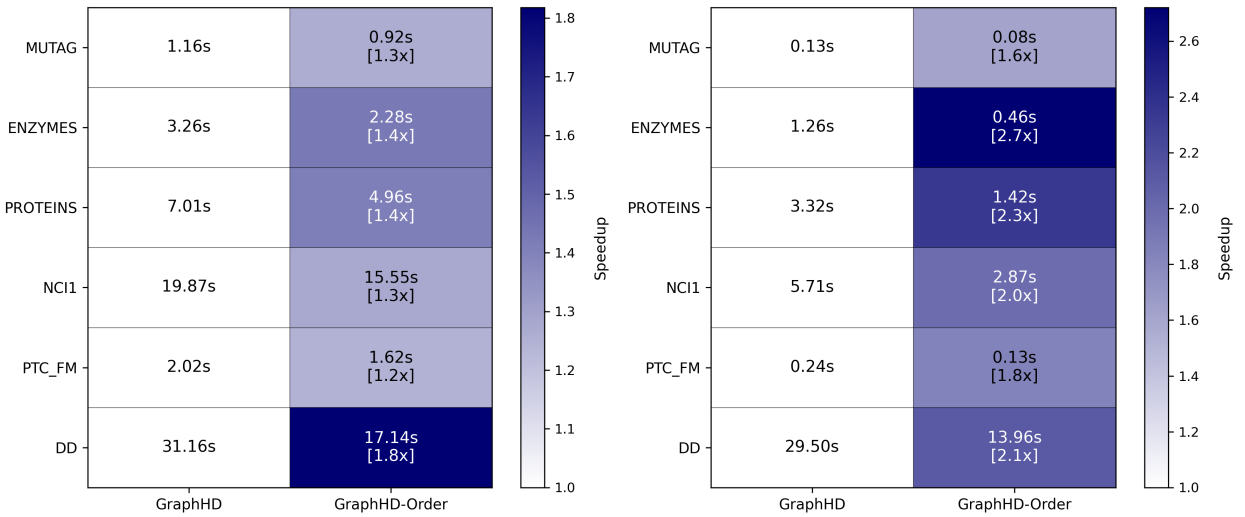


Figure 10: Mean F1-score across datasets using betweenness-based centrality metric. Results are shown for GraphHD, GraphHD-Level, and GraphHD-Order. DD dataset is omitted due to a timeout

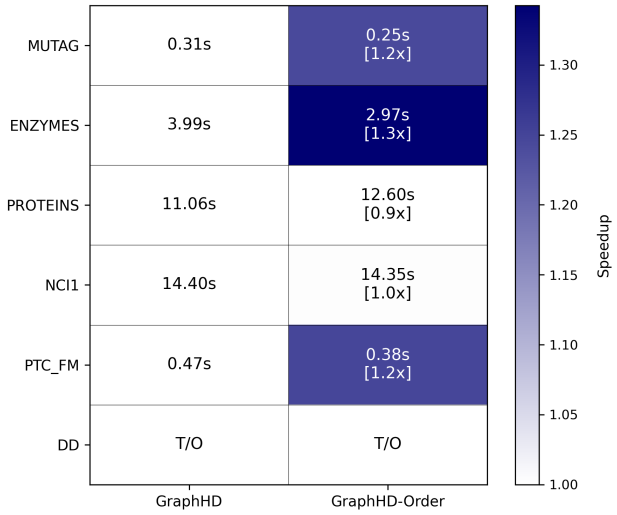
4.3 Statistical significance

Because the differences reported above are small, we complement the mean F1-scores with a paired significance analysis. Within each repetition a single random 80/20 split is shared by all methods (Section 3.5);



(a) centrality metric: PageRank

(b) centrality metric: degree



(c) centrality metric: betweenness

Figure 11: Runtime comparison (mean time in seconds) between GraphHD and GraphHD-Order across datasets for (a) PageRank, (b) degree, and (c) betweenness

we compare methods with the Wilcoxon signed-rank test over the per-repetition weighted F1-scores and apply the Holm–Bonferroni correction across the family of comparisons. We emphasize effect sizes alongside p-values, since with tens of repetitions even sub-1% F1 differences can reach significance.

4.3.1 Centrality measures.

Replacing PageRank with an alternative centrality within GraphHD changes the F1 score by a small amount—a mean of 0.010 in absolute value (at most 0.034) across the 28 dataset–centrality comparisons (Table 3). Seven are significant after Holm correction, and they concentrate on the two smallest datasets (MUTAG and PTC_FM), where a handful of graphs shifts the score; on the larger datasets (PROTEINS, NCI1, DD) no comparison is significant and every difference falls below 0.006. Importantly, the significant differences are not one-directional. On MUTAG, PageRank outperforms every alternative (degree included, by 0.017); on PTC_FM the relationship reverses and degree surpasses PageRank by an even larger margin (0.032). No centrality is therefore systematically superior as the small accuracy differences trade places across datasets rather than favoring a single measure. This is consistent with the view that the most appropriate centrality is ultimately associated with the structural phenomenon the model seeks to capture, as discussed in Section 4.4.2. Accordingly, the centrality could be seen as a task-dependent hyperparameter, and since no measure holds a consistent accuracy advantage, it can be selected on efficiency grounds (PageRank, in particular, carries no general accuracy benefit that would justify its substantially higher runtime).

4.3.2 Encoding variants.

GraphHD-Level is significantly below GraphHD in 32 of 36 dataset–centrality cells (mean −0.066), confirming that the level-based assignment does not improve classification. GraphHD-Order, by contrast, is statistically *indistinguishable* from GraphHD in 22 of 36 cells, the remaining cells showing only small differences (overall mean −0.013). Together with its lower encoding cost ($\mathcal{O}(|V|)$ rather than $\mathcal{O}(|E|)$), this supports GraphHD-Order as an efficient simplification that preserves classification accuracy (Table 4).

Table 3: Effect of replacing PageRank with the alternative centralities within GraphHD (paired Wilcoxon signed-rank with Holm–Bonferroni correction; per-repetition weighted F1 over shared 80/20 splits. |ΔF1| is the min–max mean absolute change over the available alternatives, and *leading method* is the highest-scoring centrality on each dataset; the only two datasets with a significant lead (MUTAG, PTC_FM) disagree on the winner, so no centrality is systematically superior.

Dataset	|ΔF1| vs. PageRank	significant (Holm)	leading method
MUTAG	0.009–0.034	4/5	PageRank*
PTC_FM	0.006–0.032	2/5	degree*
ENZYMES	0.001–0.016	0/5	closeness
PROTEINS	0.003–0.006	1/5	PageRank
NCI1	0.000–0.005	0/5	degree
DD	0.002–0.003	0/3[a]	PageRank
all	mean 0.010	7/28	no systematic winner

* Statistically significant lead (Holm–Bonferroni); elsewhere the differences are not significant.
[a] On DD, closeness and betweenness exceed the runtime limit, leaving three alternatives.

Table 4: Encoding variants compared with GraphHD under the paired protocol (Wilcoxon signed-rank with Holm–Bonferroni correction; per-repetition weighted F1 over shared 80/20 splits). Each variant is contrasted with GraphHD across the dataset–centrality combinations evaluated (34 of the 6×6 grid; DD omits closeness and betweenness for runtime), tallied as significantly worse, not significant (n.s.), or significantly better. The mean ΔF1 (variant − GraphHD; negative favors GraphHD) summarizes the magnitude.

variant	sig. worse	n.s.	sig. better	mean ΔF1
GraphHD-Order	12	22	0	−0.013
GraphHD-Level	32	2	0	−0.066

4.4 Discussion of the Proposed Encodings

4.4.1 Runtime and computational complexity.

The experimental results indicate that replacing PageRank with alternative centrality metrics preserves the overall discriminative power of the encoding, while substantially improving execution time. In practice, these metrics offer different computational trade-offs, with degree being the most efficient yet they all represent a notion of node importance derived from the same underlying topology. Consequently, when used within the same encoding method, they tend to yield similar classification performance while enabling faster preprocessing and overall runtime.

In addition, GraphHD-Order provides further runtime improvements because it constructs the graph representation by bundling node hypervectors, which yields an encoding cost that scales with the number of nodes ($\mathcal{O}(|V|)$) rather than the number of edges ($\mathcal{O}(|E|)$). Since $|V| \ll |E|$ in large or dense graphs, this advantage becomes particularly evident on datasets including big graphs such as DD.

Finally, the runtime differences between variants stem entirely from the *encoding* stage: classification is the same parameter-free nearest-prototype lookup for all three encoders, and its cost is negligible—on the order of 0.01 ms per graph, below 5% of the encoding time on every dataset measured. Reporting encoding time, as we do, therefore captures the computational differences that matter at inference.

4.4.2 *Accuracy trends, saturation effects, and limitations.*

An interesting accuracy trend is that GraphHD-Order consistently improves the F1-score on DD across essentially all centrality measures. A plausible explanation is the effect of superposition saturation in the original GraphHD encoding when graphs are large: bundling an extremely large number of edge hypervectors can cause the sign-projection to stabilize around an increasingly saturated prototype, thereby washing out fine-grained structural differences between graphs. This effect is attenuated in GraphHD-Order because the representation is formed by bundling node-related hypervectors, which substantially reduces the degree of superposition and helps preserve discriminative signal in the final encoding.

Although the proposed methods achieve comparable classification performance and improved encoding times over the original method, it is important to highlight a key limitation of this experimental setting: all approaches -including GraphHD- rely exclusively on structural connectivity and do not incorporate domain-specific chemical or biological attributes. While such information can, in principle, be integrated, it is strongly dependent on the dataset and the application domain (e.g., the availability and relevance of physicochemical descriptors, node/edge labels, or functional annotations). This limitation is particularly evident in the comparatively low performance obtained on the ENZYMES dataset (Table 2). In this dataset, classification depends on functionally relevant motifs and biochemical patterns that are not determined by topology alone, as enzymes are grouped according to the type of chemical reaction they catalyze. Consequently, two graphs with distinct functional behavior may still exhibit similar connectivity patterns and thus be mapped to nearly identical hypervectors, hindering discrimination when only structural information is used. A straightforward way to address this limitation in a task-specific manner is to compute additional descriptors capturing domain phenomena (e.g., physicochemical or functional features), encode them into a hypervector representation, and then fuse this information with the structural hypervector produced by the proposed methods via bundling. This strategy corresponds to a standard data fusion mechanism in hyperdimensional computing and offers a simple path to incorporate complementary, domain-dependent signals when higher accuracy is required for a particular application.

4.4.3 *Scientific significance of the design choices: what GraphHD-Order preserves and structural comparison with GraphHD*

The most notable empirical result of this study is that GraphHD-Order, which discards edge encoding entirely, is statistically indistinguishable from GraphHD on most dataset–centrality combinations (Section 4.3). Because this bears directly on what structural information the encoding actually retains, we examine the relationship between the two methods more closely.

First, after a centrality measure has been selected, both encodings can be viewed as fingerprints derived from the same centrality information. In GraphHD, every edge contributes a bound pair $H_i \otimes H_j$ formed from the hypervectors of its endpoints' centrality ranks. Because these node hypervectors are drawn from a list of independent random vectors, two edge vectors are similar only when they join endpoints with the same pair of ranks; the graph representation therefore behaves as a tally over pairs of endpoint centralities—a summary of how node-importance values are wired together. GraphHD-Order, in turn, bundles one hypervector per node, so its representation is a tally over individual centrality levels—the marginal distribution of node importance. Seen this way, GraphHD does not store the graph's topology as such: it already reduces the graph to a histogram over centrality-rank pairs. The conceptual gap to GraphHD-Order is thus not "structure versus no structure" but *joint versus marginal* of the same underlying centrality information.

The centrality metric is itself a structural feature extractor, since closeness, betweenness, Katz, and eigenvector centrality each summarize a node's multi-hop neighborhood into a single value before any edge is processed, and even degree metric encodes immediate connectivity. Much of the topological information that GraphHD re-introduces through edge binding has therefore already been folded into the node-level hypervectors bundled by GraphHD-Order. The explicit pairing of endpoints does add a finer, second-order signal—namely, the correlation between the centralities of adjacent nodes—but our paired analysis indicates that, on these benchmarks, this additional signal is largely not class-discriminative and the marginal centrality profile appears to be nearly sufficient for classification. We emphasize that the scope of this claim is comparability for the classification task, not representational equivalence.

5 Conclusion and Future Work

This work investigated the application of Hyperdimensional Computing (HDC) to graph classification by extending the original GraphHD framework in two complementary directions. First, we analyzed the effect of replacing GraphHD's default PageRank-based node assignment with alternative centrality metrics

(degree, closeness, betweenness, Katz, and eigenvector). Second, we introduced two encoding variants - GraphHD-Level and GraphHD-Order- that incorporate centrality information through distinct strategies, with GraphHD-Order additionally simplifying the algorithm by omitting edge encoding.

The experimental results show that GraphHD is largely robust to the specific centrality metric used: alternative measures proposed in this paper typically yield F1-scores close to the original PageRank configuration, while offering substantial runtime differences that can be exploited for efficiency. Among the proposed variants, GraphHD-Order emerges as the most competitive alternative to the baseline, achieving comparable (and in some cases improved) classification performance while providing consistent speedups in encoding time, especially on large graphs. To facilitate reproducibility and further experimentation, we also release an updated implementation of all methods, including command-line parameterization, CPU-level parallelization, and additional code improvements.

Overall, these findings support the potential of HDC-based models as lightweight and efficient alternatives to more complex kernel-based or neural approaches. All evaluated encodings rely exclusively on structural connectivity and do not incorporate domain-specific chemical, biological, or functional attributes of case studies. In datasets where such information is critical, topology alone may be insufficient to capture the properties required for accurate classification.

Several directions for future research follow naturally. A first path is to enrich the encoding with domain-specific attributes (e.g., physicochemical descriptors, node/edge labels, or functional annotations) and combine them with the structural hypervector via bundling, enabling a principled form of data fusion. A second direction is to integrate attention-like mechanisms into the HDC pipeline to selectively weight graph components during encoding. Finally, the algebraic and symbolic nature of HDC suggests promising opportunities beyond supervised classification, including similarity search, motif detection, and explainability analyses in scientific domains.

Supplementary Information

The implementation of scripts, datasets and detailed execution results are available at https://github.com/gustavovazquez/ 2025.

Acknowledgement

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

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] A. Zakeri, Z. Zou, H. Chen, and M. Imani, "Configurable hyperdimensional graph representation," *Artificial Intelligence*, vol. 347, p. 104384, Oct. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370225001031

[10] I. Sica and G. Vazquez, "Exploring Centrality Measures and Encoding Variants for Graph Classification in Hyperdimensional Computing," in *CLEI Conference Proceedings*, Valparaíso, 2025.

[11] 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

[12] 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

[13] 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.

[14] 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.

[15] 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

[16] 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

[17] 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.

[18] 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

[19] 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

[20] 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

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

[22] 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

[23] 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