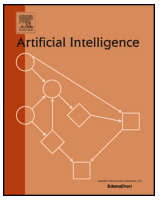
Artificial Intelligence 347 (2025) 104384

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Configurable hyperdimensional graph representation

Ali Zakeri *, Zhuowen Zou, Hanning Chen, Mohsen Imani

University of California, Irvine, CA, 92697, USA

ARTICLE INFO

Keywords:
Hyperdimensional computing
Vector symbolic architecture
Knowledge graph reasoning
Graph representation

ABSTRACT

Graph analysis has emerged as a crucial field, offering versatile solutions for real-world data representation, from social networks to biological systems. However, the intricate nature of graphs often necessitates a degree of processing, such as learning mappings to a vector space, to perform analysis tasks like node classification and link prediction. A promising approach to this is Hyperdimensional Computing (HDC), inspired by neuroscience and mathematics. HDC utilizes high-dimensional vectors to efficiently manipulate complex data structures and perform operations like superposition and association, enhancing knowledge graph representations with contextual and semantic information. Nevertheless, addressing limitations in existing HDC-based approaches to graph representation is essential. This paper thoroughly explores these methods and presents ConfiGR: Configurable Graph Representation, a novel framework that introduces an adjustable design, enhancing its versatility across various graph types and tasks, ultimately boosting performance in multiple graph-related tasks.

1. Introduction

The study of graphs and their analysis is a pivotal field that has garnered substantial attention in recent decades [1]. Graphs provide a versatile framework for representing an extensive array of real-world data, encompassing domains such as social networks [2,3] and biological networks [4]. Moreover, many practical challenges can be reframed as various graph-related tasks, including link prediction [5] and knowledge graph completion [6].

Knowledge graphs, as a distinctive category, serve as a structured and interconnected representation of knowledge, capturing relationships and semantic associations between entities. Leveraging a graph-based framework, knowledge graphs facilitate efficient information traversal and retrieval, catering to various applications like recommendation systems [7], question answering [8,9], and knowledge discovery [10]. Unlike conventional relational databases or unstructured data repositories, knowledge graphs offer a more intuitive and holistic understanding of the underlying knowledge. This, in turn, streamlines the integration of diverse data sources and enables advanced reasoning and inference capabilities [11].

Yet, the intricate nature of graphs presents challenges in extracting meaningful insights, with their non-Euclidean characteristics, further complicating matters. This has traditionally led to the adoption of strategies involving acquiring graph representations within a Euclidean space, employing embedding techniques [12,13]. These representations, once achieved, greatly facilitate diverse graph-related tasks, including node classification [14,15] and link prediction, enhancing accessibility and efficacy.

Hyperdimensional Computing (HDC), a novel paradigm inspired by concepts from cognitive neuroscience and mathematics, offers a promising approach to enhance the representation and processing of graphs. Diverging from the conventional low-dimensional

* Corresponding author.
E-mail address: azakerij@uci.edu (A. Zakeri).

https://doi.org/10.1016/j.artint.2025.104384
Received 25 September 2023; Received in revised form 31 March 2025; Accepted 7 June 2025

Available online 13 June 2025

representations employed in embedding methods, this computational framework leverages high-dimensional vectors to encode information, enabling robust and efficient manipulation of complex data structures. HDC allows for the representation of relationships through distributed vectors and supports operations such as superposition and association, which are particularly valuable for modeling graph structures.

Utilizing hyperdimensional computing provides a multitude of advantages for modeling graph structures. Firstly, it empowers the inclusion of context and semantic information into knowledge graph representations, thereby enhancing our understanding of entity relationships. This is achieved through operations such as binding and bundling, which naturally integrate node attributes and their contextual interactions—an approach supported by recent studies [16,17]. Additionally, HDC facilitates efficient similarity comparisons and information retrieval within the knowledge graph, which in turn accelerates query processing and enhances reasoning capabilities. These advantages collectively highlight the potential of HDC to capture rich semantic details, making it a promising alternative to conventional embedding methods.

To potentially further advance HDC-based graph representation methods, there is a growing interest in addressing their limitations and refining the associated techniques. While this computing framework is relatively new, researchers have begun proposing innovative approaches aimed at improving the efficiency and scalability of high-dimensional vector operations, which can be beneficial for handling large-scale graphs. Some recent efforts explore optimized algorithms, parallel computing techniques, and hardware acceleration to enhance the overall performance of hyperdimensional computing in graph representation [18].

Present HDC works in graph-related tasks, although somewhat limited in number, provide frameworks for specific applications like link prediction, general graph classification and graph matching [16,17,19]. These frameworks effectively process input graphs and perform the intended task on the generated graph representations but often incorporate fixed design structures with minimal adjustable hyperparameters, constraining their adaptability when confronted with diverse input graph types or alternative task requirements.

Focusing on HDC-based methods applied to graph-related tasks, this study seeks to provide a comprehensive overview, while also introducing the novel framework ConfiGR: Configurable Graph Representation. We thoroughly examine fundamental HDC-based graph representation techniques, categorizing them into two primary types. Subsequently, we introduce the innovative adjustable design, which can be tailored to suit specific data and task requirements. This adaptability enables the model to harness the benefits of both design types, ultimately yielding enhanced performance outcomes. Furthermore, by evaluating our method against standard knowledge graph embedding and GCN-based approaches on established benchmarks, our work underscores the complementary role of hyperdimensional encoding in advancing KG representation.

2. Hyperdimensional computing

Hyperdimensional Computing (HDC), also known as Vector Symbolic Architecture (VSA) [20–22], possesses the capacity to transform data structures into distributed representations. HDC employs a holographic approach to encode data structures in high-dimensional vector spaces, utilizing random hypervectors as foundational elements for representation. This encoding process is facilitated through the application of three fundamental operations, establishing an algebraic framework within the high-dimensional space.

In this work, we employ individual random bipolar vectors with a dimension of D, where each component is drawn from $\{-1, +1\}$, as our fundamental building blocks. The similarity between these vectors is assessed through their dot product, denoted as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \frac{1}{D} \sum_{i=1}^{D} x_i y_i$. This metric serves as an indicator of both the equivalence of atomic objects and the structural similarity among complex objects. To create structured objects, we define three HDC operations as follows:

1. The Bundling (+) operation involves component-wise addition of hypervectors, represented as $\boldsymbol{y} = \boldsymbol{x}_1 + \boldsymbol{x}_2$. This operation effectively combines the elements, functioning as a memorization process that retains information from the input data within the resultant vector. The bundled hypervector maintains similarity to its individual atomic hypervectors, as indicated by $\langle \boldsymbol{y}, \boldsymbol{x_1} \rangle \gg 0$, making it particularly suitable for representing sets.

2. The Binding ($\odot$) operation between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is performed using the Hadamard product, which involves component-wise multiplication between the two hypervectors, expressed as $\boldsymbol{y} = \boldsymbol{x}_1 \odot \boldsymbol{x}_2$. The resultant conjunct hypervector $\boldsymbol{y}$ differs from its individual vectors, as illustrated by $\langle \boldsymbol{y}, \boldsymbol{x}_1 \rangle \approx 0$. Importantly, this operation is reversible; for instance, when working with bipolar hypervectors, it is possible to retrieve $\boldsymbol{x}_1$ through element-wise multiplication of $\boldsymbol{y}$ and $\boldsymbol{x}_2$, denoted as $\boldsymbol{x}_1 = \boldsymbol{y} \odot \boldsymbol{x_2}$. Since binding effectively maintains the information of its constituents without increasing the size, it can be viewed as a form of variable binding.

3. The Permutation (ρ) operation, denoted as $\boldsymbol{y} = \rho(\boldsymbol{x})$, involves applying a cyclic shift to the components of $\boldsymbol{x}$. Permutation exhibits distributive properties over both bundling, where $\rho(\boldsymbol{x_1} + \boldsymbol{x}_2) = \rho(\boldsymbol{x}_1) + \rho(\boldsymbol{x}_2)$, and binding, where $\rho(\boldsymbol{x}_1 \odot \boldsymbol{x}_2) = \rho(\boldsymbol{x}_1) \odot \rho(\boldsymbol{x}_2)$. Similar to binding, permutation disassociates hypervectors, as evidenced by $\langle \rho^n(\boldsymbol{x}), \boldsymbol{x} \rangle \approx 0$ when $n \neq 0, n < D$. Given its reversibility and the natural order it introduces through repeated permutations within the hypervector, permutation finds utility in representing sequences and hierarchies at different levels.

These operations serve as the primary building blocks for designing various data structure representations within hyperdimensional computing. In the context of graphs, several works have focused on this data type and its applications, employing HDC as a method for data representation. For instance, Gayler and Levy [19] utilized HDC to represent graphs by binding vertices together to symbolize edges and subsequently adding the resulting vectors. Ma et al. [23] employed holographic reduced representation to map nodes

into high-dimensional space. This mapping seeks to learn the graph as a latent space without explicit graph memorization. Nickel et al. [24] introduced a method for generating graph embeddings in a vector space. Graph embedding typically involves a learning process to derive vector representations of graphs, ensuring that the vectors representing nodes correlate with the nodes' similarity within the graph. Additionally, Poduval et al. [25] introduced an encoding method for representing graph structures and defined several crucial cognitive functionalities over the encoded memory graph. Nunes et al. [17] introduced a baseline approach for graph classification using HDC and conducted evaluations on real-world graph classification problems. Zakeri et al. [26] demonstrate that the inherently distributed and interpretable nature of HDC makes it especially well-suited for capturing the nuances of knowledge graphs in cybersecurity applications.

Together, these studies underscore how HDC's symbolic and distributed encoding capabilities can be harnessed to capture both the structural and semantic relationships within graphs, or more specifically knowledge graphs, providing scalable and transparent alternatives to conventional embedding methods. However, none of the aforementioned works specifically delve into the intricacies of the representation encoding process itself. In these works, the framework is typically considered to be fixed and applicable across various tasks and datasets. This study marks the first attempt to comprehensively investigate graph encoding methods, conducting a thorough analysis of the advantages and disadvantages associated with multiple encoding schemes. Additionally, it introduces an adaptive encoding method that can be customized to suit specific data and task requirements, offering a flexible approach to encoding, and improvements across various evaluations compared to previous frameworks.

3. Hyperdimensional graph representation

Generating representations for graphs necessitates careful consideration of two critical aspects: scalability and practicality of the representation. An ideal graph representation method should maintain the quality of its output as the graph's size increases, while also providing a representation that can be seamlessly employed in various related tasks without the need for complicated preprocessing steps.

To gain a more in-depth understanding of these attributes, we can approach the issue from an encoding/decoding standpoint. In this context, the encoding process needs to possess the ability to produce precise representations while remaining adaptable to the scale of the input. This implies that even as the graph increases in size and complexity, the encoding should continue to adeptly encapsulate its fundamental characteristics without compromising its effectiveness. On the other hand, the decoding process should possess the ability to precisely reconstruct the original graph or extract any necessary features from it. A robust decoding mechanism is vital to ensure that the generated representation can be translated back into meaningful graph structures without loss of crucial information.

In the context of hyperdimensional computing, several graph representation methods have been proposed, most of which use either bundling or binding as their main approach to memorize the graph structure in hypervectors. Both approaches have their advantages over each other. Bundling allows the representation to be easy to process and decode. However, as our theoretical analyses later in the paper demonstrate, it may limit the scalability and capacity of the learned representation as the graph becomes more complex. On the other hand, binding does not impose any limitations on the representation capacity and can handle larger graphs without loss of information. However, decoding bound hypervectors can be challenging and, at times, impractical, especially when the binding process becomes more intricate. [27]

In the following sections, we will explore two distinct HDC graph representation methods: the inclusive method, primarily achieved through bundling operations, and the exclusive method, accomplished through binding. We will provide in-depth explanations and formal analyses of both the inclusive and exclusive graph representation methods, with a summary provided in Fig. 1. Understanding the strengths and limitations of each approach will enable us to make informed decisions when selecting the most appropriate method for specific graph-related tasks within the framework of hyperdimensional computing.

3.1. Inclusive graph representation in hyperspace

Although there may be variations in algorithms and the arrangement of nodes, the majority of current HDC graph representations rely on edge correspondence as a metric for assessing structural similarity. Obtaining the inclusive representation follows such scheme.

For an undirected graph denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the inclusive method initially assigns a random bipolar hypervector $\boldsymbol{H}_i$ of dimensionality D to each node $i \in V$ within the graph. Subsequently, it constructs the codebook matrix $H = [\boldsymbol{H}_1, \boldsymbol{H}_2, \cdots, \boldsymbol{H}_n]$, which is of size $D \times n$ and serves as the high-dimensional representation of the graph's nodes. Due to the random generation process, the node hypervectors exhibit a near-orthogonal quality, with minimal similarity between each pair of hypervectors, typically expressed as $\langle \boldsymbol{H}_i, \boldsymbol{H}_j \rangle \approx 0$, where $i \neq j$.

For every node denoted as j, the inclusive method generates a node memory hypervector by aggregating the set of its neighbors: $\boldsymbol{M}_j = \sum_{i \in Nbh(j)} \boldsymbol{H}_i$. The characteristics of the bundling operation imply that the similarity between the node memory and each of the bundled hypervectors is significantly greater than zero, expressed as $\forall i \in Nbh(j), \langle \boldsymbol{M}_j, \boldsymbol{H}_i \rangle \gg 0$. This property facilitates the retrieval of each neighbor during the reconstruction process through similarity assessment.

A unified hypervector for the entire graph is subsequently generated through a two-step process. Initially, the model binds each node hypervector with its respective node memory, resulting in $\boldsymbol{H}_i \odot \boldsymbol{M}_i$. This step introduces differentiation between node memories, enabling the model to correctly identify the corresponding memory for each node during the graph reconstruction. The model combines all these associated pairs into the graph hypervector $\boldsymbol{G}$ for the final step:

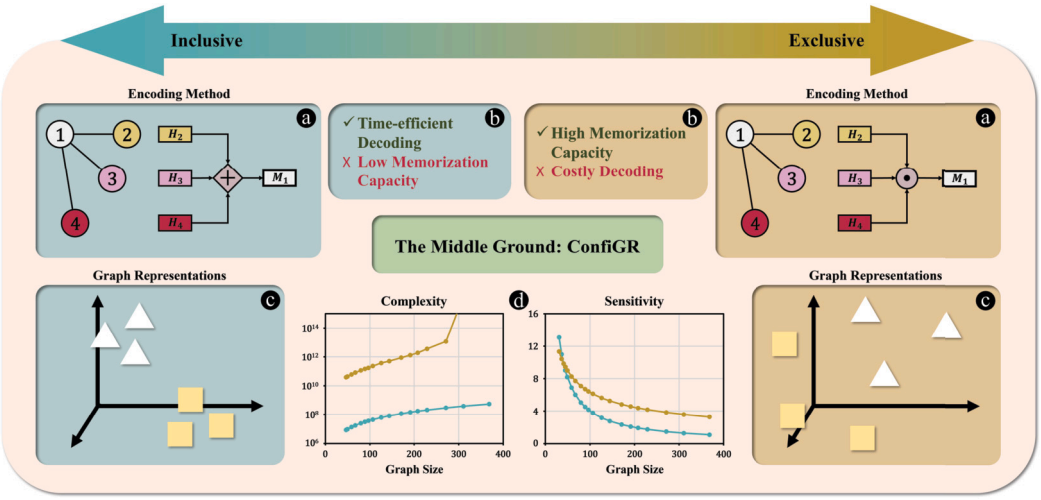


Fig. 1. Overview of the HDC-based graph representation methods, providing a comparative summary of their general properties, both inclusive and exclusive: (a) Encoding process. A key distinction between the two methods lies in how they create node memory. The inclusive approach employs the bundling operation, whereas the exclusive method utilizes binding to generate node memory. (b) Key advantages and disadvantages of each method. (c) Graph representations. The plots showcase the similarities and differences between graph hypervectors. Graphs sharing similar structures, denoted by the same shape in the plot, exhibit closer representations in hyperspace when encoded using the inclusive method compared to the exclusive approach. (d) Computation complexity and noise sensitivity plots. The values representing computation complexity indicate the number of bit computations required by each method to encode and decode graph representations, providing insights into their computational efficiency. Sensitivity serves as a metric to showcase the memorization quality of each approach.

$$\begin{aligned} \boldsymbol{G} &= \sum_{i=1}^{n} \boldsymbol{H}_i \odot \boldsymbol{M}_i = \sum_{i=1}^{n} \boldsymbol{H}_i \odot \sum_{j \in Nbh(i)} \boldsymbol{H}_j \\ &= 2 \sum_{(j,i) \in E} \boldsymbol{H}_j \odot \boldsymbol{H}_i \end{aligned} \tag{1}$$

The final equation employs the distributive property of binding over bundling. The outcome is a transparent model that has condensed all the edges within the graph, with each edge being symbolized by the binding of endpoint hypervectors.

To extract the graph's information from the graph hypervector, the inclusive decoding process initially retrieves each node's memory. This can be approximated by performing the binding operation between the graph hypervector $\boldsymbol{G}$ and the corresponding node hypervector:

$$\hat{\boldsymbol{M}}_i = \boldsymbol{G} \odot \boldsymbol{H}_i = \boldsymbol{M}_i + \sum_{j=1, j \neq i}^{n} \boldsymbol{H}_i \odot \boldsymbol{H}_j \odot \boldsymbol{M}_j \tag{2}$$

The retrieved node memory is labeled as $\hat{\boldsymbol{M}}_i$ and differs from the initial node memory hypervector $\boldsymbol{M}_i$ by including some additional components. As we perform the next and final step in the decoding process, these surplus components are discarded due to the orthogonality of hypervectors. Having estimated the node memory, we can examine the connection between nodes j and i by assessing the similarity between one node's hypervector and the node memory hypervector of the other, denoted as $\langle \boldsymbol{H}_j, \hat{\boldsymbol{M}}_i \rangle$. In cases where an edge exists between i and j, then $\langle \boldsymbol{H}_j, \hat{\boldsymbol{M}}_i \rangle \gg 0$. Conversely, when there is no edge between them, $\langle \boldsymbol{H}_j, \hat{\boldsymbol{M}}_i \rangle \approx 0$.

3.2. Exclusive graph representation in hyperspace

The exclusive approach employs a similar two-step encoding strategy, which decreases the number of terms combined together. This approach enhances the model's scalability concerning the graph's size and density. Same as the inclusive method, it initially creates a random hypervector with dimension D, denoted as $\boldsymbol{H}_i$, for each node.

With the required vectors created to represent individual nodes, the graph encoding can now be divided into two levels of memorization, mirroring the inclusive method. In the initial level, the method captures node information locally by associating all the neighbors for each node with each other to create a single node memory hypervector $\boldsymbol{M}_i$:

$$\boldsymbol{M}_i = \prod_{j \in Nbh(i)} \boldsymbol{H}_j \tag{3}$$

where $\prod$ represents the binding operation applied to multiple hypervectors. In the second encoding level, all the information accumulated within the node memories is bundled together in the final graph hypervector:

$$\boldsymbol{G} = \sum_{i=1}^{n} \boldsymbol{H}_i \odot \rho(\boldsymbol{M}_i) \tag{4}$$

where a permutation is employed on the node memories to entirely differentiate between the two encoding levels.

The reconstruction process for the exclusive approach can be likewise partitioned into two stages. Initially, an approximation of the node memory for each node is recovered from the graph hypervector:

$$\hat{\boldsymbol{M}}_i = \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{H}_i) = \boldsymbol{M}_i + \sum_{j=1, j\neq i}^{n} \rho^{-1}(\boldsymbol{H}_i \odot \boldsymbol{H}_j) \odot \boldsymbol{M}_j \tag{5}$$

The result of this computation is denoted as $\hat{\boldsymbol{M}}_i$, which differs from the original node memories and may contain noise terms. The subsequent step involves identifying the neighbors of each node from its node memory hypervector. As the model creates node memories through binding, the factorization problem may entail two or more factors, leading to an exponential increase in the potential solution space. The resonator network, as proposed in Frady et al. [27], Kent et al. [28], and subsequently applied in various applications [29,30], aims to address this challenge without the need for an exhaustive search through all possible factor combinations.

To achieve this, the resonator network sets an initial estimate for each factor by applying superposition to all potential vectors from the codebook. The algorithm then iteratively refines these estimates and derives new approximations based on the previous iteration's results. In this context, the algorithm utilizes the following equation to update the estimate for j^{th} factor $\hat{\boldsymbol{H}}_j$ of $\hat{\boldsymbol{M}}_i$ for the t^{th} iteration:

$$\hat{\boldsymbol{H}}_j(t+1) = f(H\bar{H}^T(\hat{\boldsymbol{M}}_i \odot \prod_{k=1, k\neq j}^{m} \hat{\boldsymbol{H}}_k(t))) \tag{6}$$

It accomplishes this by linking all previous factor estimates with the composite vector (i.e., the retrieved node memory) and subsequently calculating a weighted linear combination of codebook entries to generate the new estimate for the current factor. It's important to note that the number of factors used to construct each node memory is assumed to be equal to m, representing the maximum node degree. Additionally, f denotes an activation function utilized to keep the hypervectors bipolar.

The resonator network iteratively updates its factor estimates, progressing from the first factor to the last and then returning to the first for a new round of estimations. The iterations continue until all factors have reached a stable state, where the new estimates for each factor no longer change compared to the previous iteration. This process must be conducted for each node in the model to obtain the neighbors for all nodes, thereby reconstructing the entire graph.

4. Configurable graph representation

After examining the inclusive and exclusive graph representation models, we will introduce a novel framework that seeks to combine the strengths of both approaches. By effectively utilizing both binding and bundling operations, this new model aims to achieve enhanced scalability and reduced complexity compared to the previously discussed methods.

4.1. ConfiGR design

In our framework, we incorporate several new features that significantly enhance its flexibility, enabling the model to be tailored for specific types of graphs based on their structural parameters. The formal derivation of the representation follows a process similar to the previous frameworks, beginning with the calculation of memory hypervectors and subsequently deriving the graph hypervector. An overview of the full encoding framework is shown in Fig. 2.

One significant enhancement in our new framework involves the grouping of nodes. This grouping process can be accomplished using various methods. However, for the purposes of our study, we will maintain a straightforward approach, randomly assigning nodes to groups in a manner that ensures each group contains an equal number of nodes or, with at most a difference of one node between groups. Irrespective of how the nodes are grouped, we will leverage these groups to enhance the construction of node memory.

Suppose the nodes are separated into distinct groups represented as $S = \{s_1, s_2, \ldots, s_d\}$. We can formally define the node memory as follows:

$$\boldsymbol{M}_i = \prod_{j=1}^{d} \sum_{k\in Nbh(i), k\in s_j} \boldsymbol{H}_k \tag{7}$$

Additionally, to encode directed labeled graphs, we can extend the node memory calculation by incorporating both edge attribute and direction through a combination of binding and permutation operations. Let $\boldsymbol{R}_{ik}$ denote the hypervector corresponding to the directed edge e_{ik} from node i to node k. Then, we can express the node memory representation as:

$$\boldsymbol{M}_i = \prod_{j=1}^{d} \sum_{k\in Nbh(i), k\in s_j} \hat{\boldsymbol{R}}_{ik} \odot \boldsymbol{H}_k \tag{8}$$

where

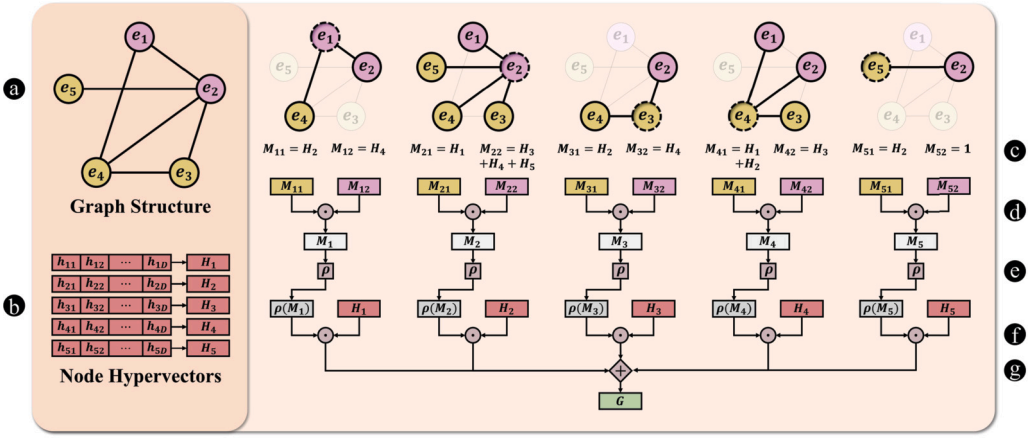


Fig. 2. An overview of the hyperdimensional graph representation framework in ConfiGR, using a sample graph containing 5 nodes and 6 edges as an example. The encoding process involves the following steps: (a) Nodes are randomly assigned to two distinct groups, as indicated by their colors. (b) Hypervectors, represented by $\boldsymbol{H}_i$ for each node, are generated. (c) Memory hypervectors are computed for each group by focusing on a single node and bundling the neighboring nodes within that group. (d) Node memories are formed by binding all their respective group memories. (e) The node memories are subjected to a permutation operation. (f) They are then associated with their corresponding node hypervectors.(g) Finally, all these components are bundled together to create the final graph hypervector, denoted as $\boldsymbol{G}$.

$$\hat{\boldsymbol{R}}_{ik} = \begin{cases} \boldsymbol{R}_{ik}, & \text{if } e_{ik} \text{ is an outgoing edge.} \\ \rho(\boldsymbol{R}_{ik}), & \text{if } e_{ik} \text{ is an incoming edge.} \end{cases} \tag{9}$$

Rather than solely relying on bundling or binding all neighbors together, we adopt a combination of both operations to create the node memory. Through mathematical analysis, we will demonstrate how this hybrid approach enhances the quality of our framework.

In the subsequent step, we bundle all the node memories together, and then permute and associate them with their respective node hypervectors. This binding process ensures that node memories can be uniquely recovered, while permutation guarantees a clear separation and independence between the two steps. The final graph hypervector and representation are derived as follows:

$$\boldsymbol{G} = \sum_{i=1}^{n} \boldsymbol{H}_i \odot \rho(\boldsymbol{M}_i) \tag{10}$$

Through the encoding process, the model generates a comprehensive representation for the input graph. However, to effectively utilize this representation, many tasks require decoding the graph hypervector and extracting valuable information about the graph's structure from its representation.

The initial step in the decoding process involves recovering the node memories. This is achieved as illustrated below:

$$\hat{\boldsymbol{M}}_i = \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{H}_i) = \boldsymbol{M}_i + \sum_{j=1, j\neq i}^{n} \rho^{-1}(\boldsymbol{H}_i \odot \boldsymbol{H}_j) \odot \boldsymbol{M}_j \tag{11}$$

where first term is node memory hypervector and the latter cover additional terms, which will be disregarded due to the orthogonality of hypervectors and, more specifically, the permutation used during the encoding.

The subsequent step involves recovering the neighbors for each node by utilizing their memory hypervectors. This process enables us to reconstruct the entire graph, including all node connections. To achieve this, we begin by distinguishing the node groups from one another and then identifying the relevant nodes within each group. Mathematically, this is accomplished by calculating the hypervectors that are bound together to form the node memory, with one hypervector for each node group. We can execute this procedure by employing the resonator network, as depicted below:

$$\hat{\boldsymbol{H}}_j(t+1) = f(H\bar{H}^T(\hat{\boldsymbol{M}}_i \odot \prod_{k=1, k\neq j}^{d} \hat{\boldsymbol{H}}_k(t))) \tag{12}$$

Once we have retrieved the memory for each group, the next step involves utilizing similarity checks to determine the nodes that belong to each specific group. The process is illustrated below:

$$\begin{cases} \langle \boldsymbol{H}_j, \hat{\boldsymbol{M}}_i \rangle \gg 0 \Rightarrow j \in Nbh(i) \\ \langle \boldsymbol{H}_j, \hat{\boldsymbol{M}}_i \rangle \approx 0 \Rightarrow j \notin Nbh(i) \end{cases} \tag{13}$$

With the completion of the decoding process, we have successfully reconstructed the entire graph from its previously learned representation.

4.2. Motivation

In this section, we delve into the foundational concepts that underpin our proposed framework. We commence by analyzing the concept of bundling and its inherent constraints regarding scalability. Subsequently, we extend our discussion to encompass the attributes of the binding operation. These operations have played pivotal roles in prior HDC-based graph representation approaches, shaping numerous properties and characteristics of these methods, all of which we comprehensively explore throughout this study.

To better understand the potential of memorization in HDC, we introduce the concept of "Capacity." Capacity quantifies the maximum number of orthogonal vectors that can be memorized through bundling in a single hypervector while maintaining full point resolution, meaning that all points can be accurately decoded back. We present a formal definition for capacity as below:

Definition. Assume the hypervector $\boldsymbol{z}$ is built from the memorization of atomic hypervectors $\{\boldsymbol{x}_i\}_{i\in[1,n]}$. The capacity of the memory hypervector is defined as the maximum number of atomic hypervectors n, such that for an arbitrary atomic hypervector $\boldsymbol{y}$, and the threshold value ϵ:

$$Pr(\langle \boldsymbol{z}, \boldsymbol{y}\rangle > \langle \boldsymbol{z}, \boldsymbol{x}_i\rangle) < \epsilon \tag{14}$$

To formally establish the memory capacity when constructing it through bundling, let's consider a scenario where we have a codebook X containing n hypervectors, each with a dimension of D. This codebook is generated in a manner where the correlation between any two hypervectors follows a Gaussian distribution: $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \sim \mathcal{N}(\mu, \sigma^2)$, where $0 \le \mu < 1$. Given these assumptions, the subsequent theorem outlines the capacity of a memory formed using the bundling operation:

Theorem 1. *Assume n hypervectors of dimension D, creating the codebook $X = \{\boldsymbol{x}_i\}_{i\in[1,n]}$. The codebook is generated in a way that correlation between each two hypervectors comes from a Gaussian distribution: $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \sim \mathcal{N}(\mu, \sigma^2), 0 \le \mu < 1$. With the similarity threshold denoted as ϵ, the memorization capacity m_c is calculated from the following equation:*

$$\epsilon = \Phi\left(-\frac{\mu m_c + 1 - \mu}{\sqrt{(\frac{1}{D} + \sigma^2)m_c - \sigma^2}}\right) \tag{15}$$

where Φ is the cumulative distribution function (CDF) of the Gaussian distribution. In the case of random orthogonal codebook hypervectors ($\mu = 0, \sigma^2 = \frac{1}{D}$), the final value for m_c is calculated as below:

$$m_c = \frac{D}{2(\Phi^{-1}(\epsilon))^2} + \frac{1}{2} \tag{16}$$

Proof. See proof in Appendix A. □

As demonstrated above, both the capacity and the quality of memorization are contingent upon the characteristics of the codebook. In the example provided, the memorization process was executed via the bundling operation, a common practice in the majority of HDC-based algorithms.

To assess the quality of memorization in ConfiGR compared to previous representation approaches, we can examine how distinct the reconstructed memories are between neighboring and non-neighboring nodes. In essence, greater distinctness indicates more accurate memorization. Fig. 3 illustrates this analysis for both the inclusive scheme and ConfiGR across various numbers of groups. The results clearly show a notably higher margin for error in the inclusive approach compared to ConfiGR, as evident in both the reconstruction outcomes and the similarity distributions.

The distributions represent the similarity between the memory and existing patterns (i.e., hypervectors for neighboring nodes) and compare it with the similarity to non-existent patterns. As anticipated, these distributions exhibit Gaussian shapes, centered around 1 and 0, respectively. However, the variance of these distributions differs depending on the method and its parameters. Specifically, the number of node groups impacts the variance in the ConfiGR results. The choice of this parameter can be tailored based on the graph's structure and parameters to strike a balance between distinct similarity distributions and computational efficiency, which we will thoroughly analyze in Section 5.

The aforementioned outcome can also be theoretically linked to the memorization capacity. By dividing neighbors into groups of bundled memories, as opposed to forming a single memory by bundling them all together, the model remains within its capacity limit. This approach results in higher-quality memorization and more precise reconstructions.

Having reviewed certain traits of previous HDC-based representations that inspired the concept of adaptive graph representation, we will broaden our analysis beyond just the bundling operation. We can dissect each HDC framework into two fundamental operations: bundling and binding. Through an examination of the encoding and decoding procedures for both of these operations, we can gain deeper insights into how distinct representation schemes contrast with one another across various aspects.

Bundling, achieved through summation, can be reversed using similarity measurements. If an element from the codebook, from which the constituents are chosen for bundling, exhibits a sufficiently high similarity with the memory hypervector, it is safe to conclude that the element is part of the memory. This process is relatively straightforward; the bit computation needed for decoding

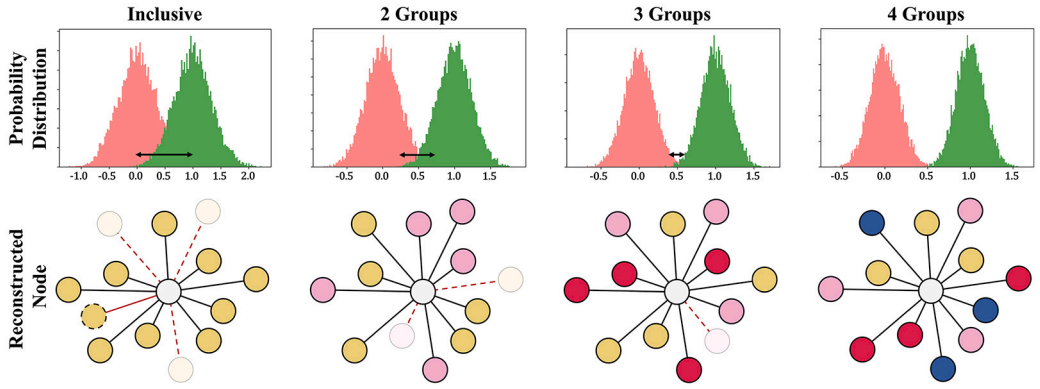


Fig. 3. The plots depict similarity distributions and an example of a reconstructed graph using various encoding models. Dashed red edges highlight the absent edges in the reconstructed graph, while the solid red edge represents an erroneously reconstructed edge to a non-adjacent node. When the number of groups is increased, which is visually represented by different node colors, there is a reduction in the number of nodes bundled within each group. Consequently, this increases the separation between the distributions of existing and non-existing patterns, as indicated by the arrows on the distributions. This results in improved decoding accuracy. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

a memory constructed from a codebook containing n hypervectors, each with D dimensions, is $O(nD)$; the similarity check is necessary between the memory hypervector and each of the n codebook elements.

However, the binding operation, which utilizes association, may entail a more intricate process for reversal. Since binding transforms the atomic hypervectors into a different space than the original codebook, recovering the constituents necessitates searching this target space. To formalize, when binding is conducted over n distinct hypervectors, selected from a codebook consisting of m hypervectors, the search space for the bound hypervectors is as large as $\binom{m}{n}$. As mentioned earlier, resonator networks currently provide the most feasible approach for conducting this search. This framework initiates its search by choosing a point within the target space and progressively refining its estimations for the constituents while ensuring that the search remains within the target space. Despite the approximation frameworks avoiding the computationally expensive exhaustive search across the target space, they are still intricate algorithms that involve numerous operations, including vector-matrix multiplications. Moreover, many of these frameworks require multiple iterations, introducing further complexity and execution time.

When comparing these two operations, it becomes evident that the decoding process for a memory constructed using binding, as in the exclusive method, while not encountering the same memorization quality issues as the inclusive approach and exhibiting significantly superior memorization accuracy, can be substantially more intricate and resource-intensive in contrast to the use of bundling. By integrating both analyzed operations in ConfiGR, we can strike a balance between the two encoding methods, potentially achieving the advantages of both. This allows us to expand the limits and capacity for information storage while maintaining decoding complexity at a manageable level.

5. Theoretical and empirical analyses

In this section, we will present the experiments conducted on the ConfiGR framework and delve into their significance. Our results encompass three main categories. First, we perform a theoretical analysis of ConfiGR's mathematical properties, which is further divided into an assessment of representation quality and an evaluation of framework complexity. These analyses collectively demonstrate the effectiveness of the ConfiGR framework by elaborating on its characteristics. Second, we showcase the practical effectiveness of ConfiGR in various graph-related tasks, including graph reconstruction. We empirically evaluate our model by quantifying multiple performance metrics across a range of input data and model hyperparameter configurations. Throughout this section, we examine both fully inclusive and exclusive graph representation models, as well as different ConfiGR models with varying numbers of incorporated groups in the encoding process. Finally, we assess the model's performance in the knowledge graph link prediction task. We measure our results using established datasets and compare them to several recent studies tackling the same task.

5.1. Noise sensitivity analysis

A key aspect of any representation method lies in the quality of the outcomes it generates, which are in the form of hypervectors in HDC-based models. In this scenario, we can measure the precision of the vectors stored within the memory hypervectors by assessing the sensitivity of the main vector compared to the noise, formally defined as:

Definition *(Sensitivity).* The sensitivity, also referred to as noise sensitivity, for distinguishing between an existing pattern $\boldsymbol{x}$ and a non-existent pattern $\boldsymbol{x'}$ within the hypervector $\boldsymbol{z}$ can be formally defined as follows:

$$S := \frac{\mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{z} \rangle] - \mathbf{E}[\langle \boldsymbol{x'}, \boldsymbol{z} \rangle]}{\mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{z} \rangle]} \tag{17}$$

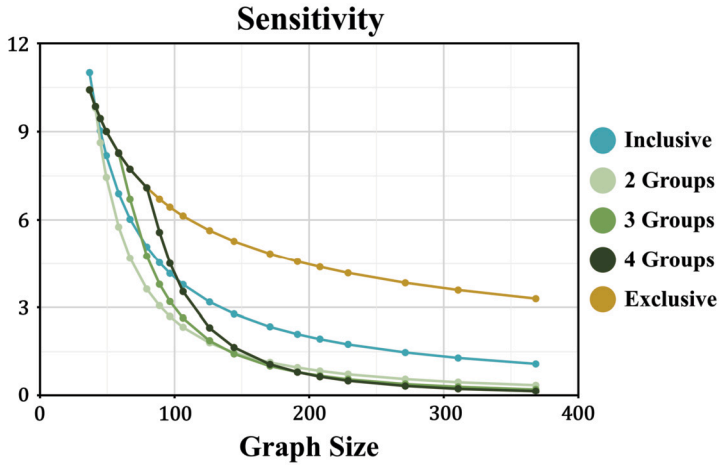


Fig. 4. Comparison of sensitivity among exclusive, inclusive, and a range of ConfiGR models. Sensitivity is computed concerning the decoding of the reconstructed node memory, with graph size as the variable. We maintain a constant hypervector dimensionality and graph density, set at $D = 4000$ and density $= 0.05$, respectively.

where $\mathbf{E}[.]$ and $\mathbf{std}[.]$ denote the expectation and standard deviation over the distributions of $\boldsymbol{x}, \boldsymbol{x'}$.

A more comprehensive explanation of this definition is available in Frady et al. [31]. When utilizing bundling for storage, the sensitivity for detecting each component in the memory hypervector diminishes as additional hypervectors are incorporated. To demonstrate this, and quantify the effectiveness of graph representation in each approach, we can compute sensitivity values for the reconstructed memories. This enables us to illustrate the distinctions in memorization capability and capacity among the methods.

Theorem 2. *For the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$, $|\mathcal{E}| = e$, and the maximum degree is m, the sensitivity of the reconstructed node memory hypervectors for the inclusive scheme, exclusive scheme, and ConfiGR can be calculated using the following equations, respectively:*

$$
\begin{aligned}
S_1 &= \frac{1}{\sqrt{e}}\sqrt{D} \\
S_2 &= \frac{1}{\sqrt{n}}\sqrt{D} \\
S_3 &= \begin{cases} \frac{1}{\sqrt{n(\frac{m}{d})^d}}\sqrt{D}, & m > d \\ \frac{1}{\sqrt{n}}\sqrt{D}, & m \leq d \end{cases}
\end{aligned}
\tag{18}
$$

where d is the number of groups used in the ConfiGR representation.

Proof. See proof in Appendix A. □

Fig. 4 illustrates a sensitivity comparison among the inclusive and exclusive methods, along with ConfiGR models employing various numbers of groups, across different graph sizes. Generally, the exclusive method exhibits the highest sensitivity, except for smaller graphs with fewer edges, where the inclusive method can slightly outperform it. The ConfiGR models don't surpass the exclusive method, as can also be inferred from Equation (18). In comparison with the inclusive method, ConfiGR models can display both higher and lower sensitivities, depending on the graph parameters and the number of groups. Additionally, employing more groups for smaller graphs tends to increase sensitivity for ConfiGR, while for denser graphs, it may introduce more noise. It's important to note that the figure focuses on scenarios involving dense graphs, where graph density remains fixed throughout the experiment.

5.2. Computation complexity analysis

It's essential to assess the computational requirements for each model during the reconstruction process. While all models share a similar computational order for encoding graph representations, the decoding process differs in complexity among the three models. This disparity can be quantified as follows:

Theorem 3. *For the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$, and the maximum degree is m, the bit operations required to decode the high-dimensional graph representation using the inclusive scheme, exclusive scheme, and ConfiGR can be calculated with the following equations, respectively:*

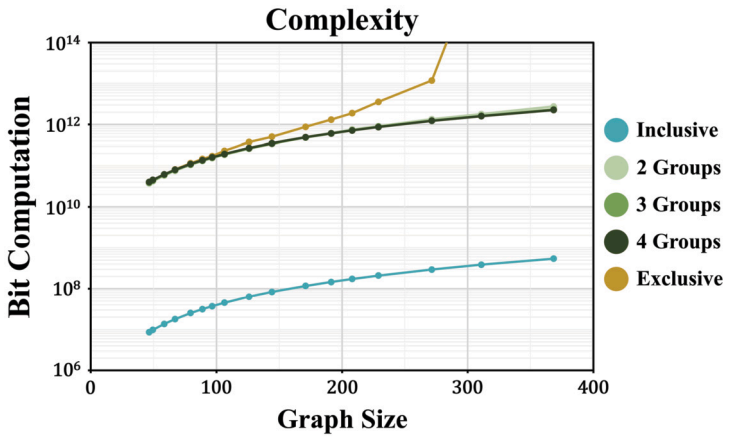


Fig. 5. Comparison of computation complexity among exclusive, inclusive, and a range of ConfiGR models. The complexity values, calculated in bit computation, are measured for decoding the reconstructed node memory as a function of graph size. We maintain a constant hypervector dimensionality and graph density, set at $D = 4000$ and density $= 0.05$, respectively.

$$\begin{aligned} T_1 &= O(n^2 D) \\ T_2 &= O(n^2 D^2 (1 + \frac{m}{n} I_{n,m})) \\ T_3 &= O(n^2 D^2 (1 + \frac{d}{n} I_{n,d})) \end{aligned} \tag{19}$$

where d is the number of groups used in the ConfiGR representation, and $I_{n,k}$ is the number of iterations required for the resonator network to converge when decomposing k factors with a codebook size of n.

Proof. See proof in Appendix A. □

Fig. 5 presents the computational complexity measurements, computed using Equation (19). The ConfiGR values demonstrate an almost quadratic increase concerning the graph size utilized for reconstruction, given fixed graph density values. The exclusive model follows a comparable pattern until it reaches a point where the complexity escalates infinitely as the graph size increases, rendering the model incapable of completing the task. In contrast, the complexity of the inclusive method increases quadratically under comparable conditions, with its values being smaller than those of ConfiGR by a factor of dimensionality.

Additionally, these trends can be explained by Theorem 3, in which ConfiGR's complexity is denoted T_3. Within the range of group sizes d used in our experiments, the term $\frac{d}{n} I_{n,d}$ remains relatively small, so it does not alter the overall order of complexity. However, if d increases substantially—thereby expanding the number of node groups—the resonator network's iteration count can grow exponentially, as clearly observed in the exclusive encoding (where $d = n$). This insight clarifies why the exclusive approach becomes unable to complete the task for large graphs, while ConfiGR maintains feasible performance despite the nearly quadratic growth in complexity. Moreover, this theoretical analysis informed our choice of group sizes during the empirical experiments, ensuring a balance between improved performance and tractable computation.

In summary, the qualitative analyses underscore the effectiveness of ConfiGR in terms of representation quality and scalability, in comparison to both inclusive and exclusive graph representation schemes. Our framework overcomes the practicality challenges arising from extensive computation in exclusive methods, while maintaining a reasonable level of memorization quality, often surpassing the performance of the inclusive method. These enhancements are complemented by the newfound flexibility of the design, a feature lacking in both of the previous methods. This adaptability empowers users to configure the design for optimal performance on specific graphs, tailored to their unique structures and properties, as well as managing the trade-off between model performance and computational constraints.

5.3. General graph results

Additionally, we conduct a series of experiments to comprehensively assess our model, supplementing the theoretical evaluations outlined earlier. When addressing the graph reconstruction task, ConfiGR, along with the other examined graph representation schemes, can be viewed as classifiers. In this context, each edge within a graph can be assigned a label, either "1" denoting an "existing edge" or "0" signifying a "non-existent edge" in comparison to the complete graph of the same size. Consequently, the goal of the graph reconstruction task revolves around classifying the edges within the graph, permitting the evaluation of these models using classification performance metrics.

In this section, the experiments utilize graphs generated by a random graph generator. This generator uniformly selects instances from the pool of all potential graphs, considering a defined size and density. The number of groups for ConfiGR's encoding is set

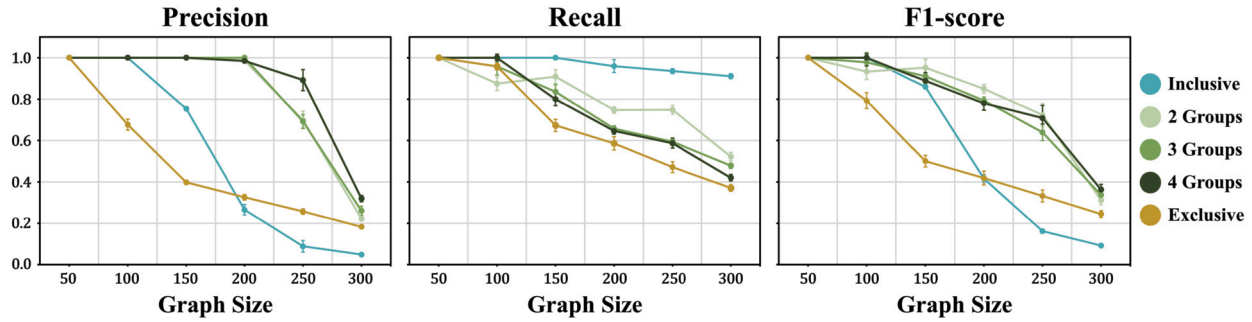


Fig. 6. Graph reconstruction accuracy plot for ConfiGR and previous methods, for different graph sizes. The dots represent mean values, while the error bars indicate standard deviations calculated from 10 repeated experiments. We maintain a constant hypervector dimensionality and graph density, set at $D = 4000$ and density $= 0.05$, respectively.

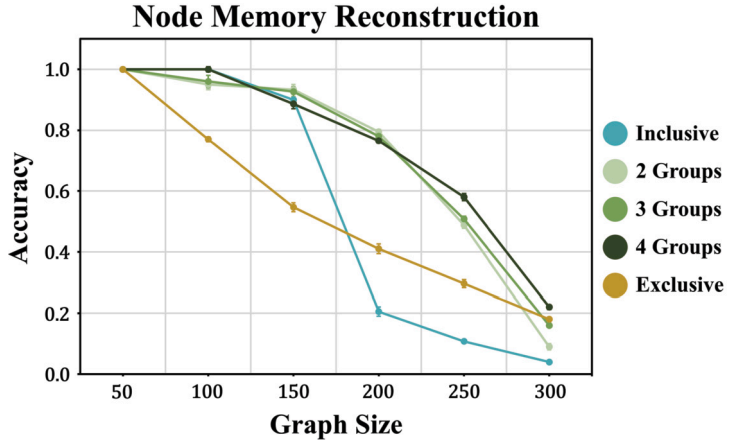


Fig. 7. Node reconstruction accuracy plot for ConfiGR and previous methods. Accuracy represents the percentage of nodes with perfectly reconstructed memory. The lines represent mean values, while the error bars indicate standard deviations calculated from 10 repeated experiments. We maintain a constant hypervector dimensionality and graph density, set at $D = 4000$ and density $= 0.05$, respectively.

to $d = 2, 3, 4$, covering a meaningful range of complexity while remaining computationally feasible. Additionally, to enhance result accuracy, the experiments were conducted multiple times, yielding both averaged performance values and accompanying standard deviations.

Fig. 6 presents the classification performance metrics for ConfiGR when performing the reconstruction task with varying dimensionalities and graph sizes. The results indicate the effectiveness of our model across all three group sizes; ConfiGR demonstrates strong precision and recall values across different graph sizes. Notably, while the inclusive method achieves the highest recall values, it struggles with the precision, caused by a significant number of false positives in its classification results. On the other hand, the exclusive method performs less effectively than ConfiGR across all metrics.

Moreover, Fig. 7 illustrates the classification performance metrics for the representation methods when reconstructing node memory across various graph sizes. This experiment focuses exclusively on node-level reconstruction, rather than full graph reconstruction. A node memory is deemed accurately reconstructed when it effectively encompasses neighboring nodes without introducing erroneous connections. The results highlight ConfiGR as the top performer, with the specific group size yielding the best results depending on the graph parameters. The inclusive method keeps pace with ConfiGR for smaller graphs but exhibits significantly inferior performance as the graph size increases. Meanwhile, the exclusive approach generally lags behind in performance across most graph sizes.

Furthermore, we evaluated the efficiency of our model in solving the shortest path problem—a fundamental graph operation with applications ranging from network routing to transportation systems [32]. In our approach, the hyperdimensional encoding is first decoded to reconstruct the graph structure, allowing us to obtain a binary connectivity map of the graph. We then apply the Dijkstra's algorithm, a standard shortest path algorithm, to this reconstructed graph to determine the optimal route between any two nodes.

Fig. 8 demonstrates that ConfiGR models outperform both the inclusive and exclusive methods in accurately navigating the reconstructed graph to identify the optimal paths. Among the ConfiGR configurations, the model with 4 groups achieves the highest success rate, while the 2-group and 3-group variants yield similar outcomes, with the 2-group model slightly outperforming the 3-group model. These findings illustrate that our hyperdimensional encoding effectively recovers graph connectivity, thereby supporting efficient computation of shortest paths.

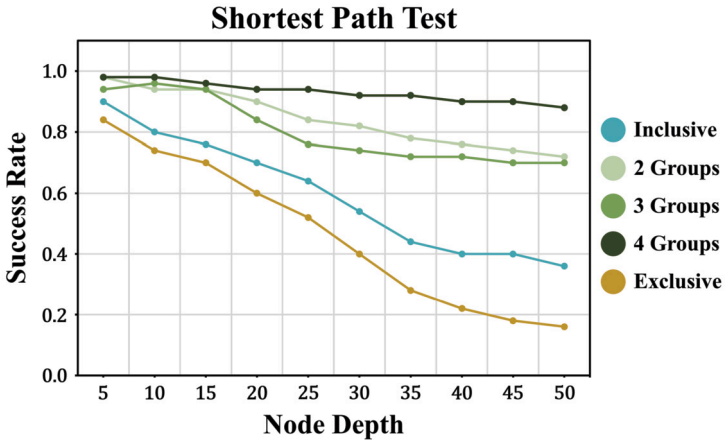


Fig. 8. Graphical representation displaying the outcomes of the shortest path test. The horizontal axis represents the distance between two nodes, while the vertical axis showcases the success rate achieved by the models during the test. The experiments were conducted on a graph with a size of 15,000 nodes and a density of 0.05. These experiments were repeated 50 times, with the final success rate indicating the proportion of successful test runs. It's worth noting that the hypervector dimensionality was consistent across all the models, set at $D = 2000$.

The quantitative results collectively illustrate the enhancements that ConfiGR introduces across a range of graph-related tasks in comparison to the inclusive and exclusive methods. Moreover, the selection of node group sizes can prove to be advantageous in different scenarios, as it influences model performance. As observed in our experiments, each of the choices we present performs optimally in specific graph sizes. In general, using larger group sizes tends to yield slightly improved performances as the graph size increases. However, it's crucial to consider computational complexity, as excessive group numbers can lead to increased computation and decreased decoding quality. In our study, we ensured that the number of groups remained at a moderate number to strike a balance between enhanced performance and efficient decoding.

5.4. Link prediction results

The task of link prediction represents a fundamental challenge within the realm of knowledge graph research. Its primary objective is to forecast absent or potential connections between entities by leveraging the available graph structure. Given that knowledge graphs frequently exhibit incomplete information due to the vastness of real-world domains, link prediction plays a vital role in deducing and filling in the gaps, thereby augmenting the comprehensiveness of the graph. Effective link prediction holds significant implications for diverse applications such as recommendation systems, information retrieval, and completion of knowledge bases [33].

Much like Graph Neural Networks (GNNs), which have gained significant attention and shown remarkable performance in various graph-related tasks, ConfiGR can serve as a valuable tool for encoding the structural information of knowledge graphs in the context of link prediction. Our overarching architecture for this task, which bears resemblance to the one employed in Vashishth et al. [34], employs ConfiGR as the front-end encoder to capture essential graph structure details. Subsequently, it utilizes a scoring function, specifically DistMult [35], as the decoder to make predictions regarding missing entities.

To offer a deeper understanding of the ConfiGR methodology for knowledge graph tasks, it consists of two fundamental phases, each serving a vital function in the comprehensive process of reasoning and learning within knowledge graphs. In the first phase, the knowledge graph is initialized with low-dimensional embeddings for every entity and relation. These embeddings are then converted into hypervectors and undergo additional processing via the ConfiGR memorization procedure for directed labeled graphs, resulting in the generation of node memory hypervectors. This stage amplifies the representation of entities and relations, preserving their distinct associations and attributes within a higher-dimensional space.

The node memory hypervectors, combined with the relation hypervectors, are subsequently input into a scoring function designed to evaluate the credibility of each triple occurrence within the knowledge graph. This process provides valuable information regarding the probability of specific relationships. As the gradient-based scoring process iterates, the entire model consistently enhances the low-dimensional embeddings, resulting in progressive refinements for knowledge graph tasks. Due to HDC's intrinsic symbolic characteristic, we exclusively update the entity and relation embedding vectors, while keeping the encoding base hypervectors unchanged throughout the training process.

To evaluate ConfiGR's link prediction performance and assess its scalability, we conducted experiments on widely used benchmark datasets, including FB15K-237 and WN18RR. FB15K-237 refines the original FB15K dataset by removing inverse relations, which provides a more robust evaluation setting, while WN18RR—derived from WordNet—emphasizes semantic consistency with fewer but more meaningful relations, thereby introducing additional complexity. Table 1 summarizes the key statistics of the benchmark datasets, including the number of entities, relations, and the distribution of train, validation, and test triples. In our experiments, the original embedding vectors have a dimensionality of 400 bytes and the encoded hypervectors are set to 2000 bytes. We report standard evaluation metrics—Mean Reciprocal Rank (MRR) and Hits@k—to enable a comprehensive comparison across different

Table 1
Statistics for the selected knowledge graph datasets. The figures under train, valid, and test columns represent the number of triples in each dataset split.

Dataset	Entities	Relations	Train	Validation	Test
FB15k-237	14541	237	272115	17535	20466
WN18RR	40943	11	86835	3034	3134

Table 2
Link prediction performance of ConfiGR and previous knowledge graph embedding models on FB15K-237 and WN18RR datasets. The best results across all methods are highlighted in bold, while the top results among the HDC-based methods are underlined. ◇: Results are taken from Li et al. [36].

	FB15K-237				WN18RR			
	H@10	H@3	H@1	MRR	H@10	H@3	H@1	MRR
TransE [37]◇	0.441	0.376	0.198	0.279	0.532	0.441	0.043	0.243
DistMult [35]◇	0.446	0.301	0.199	0.281	0.504	0.470	0.412	0.444
R-GCN [38]◇	0.300	0.100	0.181	0.164	0.207	0.137	0.080	0.123
ConvE [39]◇	0.497	0.341	0.225	0.312	0.531	0.470	0.419	0.456
CompGCN [34]◇	0.535	0.390	0.264	0.355	0.546	0.494	0.443	0.479
HittER [40]◇	0.558	0.409	0.279	0.373	**0.584**	**0.516**	**0.462**	**0.503**
ComplEx-N3-RP [41]	**0.568**	**0.425**	**0.298**	**0.388**	0.578	0.505	0.443	0.488
Inclusive HDC	0.520	0.382	0.266	0.353	0.540	0.485	0.436	0.465
Exclusive HDC	0.476	0.336	0.208	0.294	0.499	0.442	0.396	0.431
ConfiGR (Proposed Method)	0.539	0.396	0.276	0.365	0.548	0.497	0.449	0.468

models. Here, MRR measures the average inverse rank of the correct entity, and Hits@k represents the percentage of queries for which the correct entity is ranked within the top k positions.

The results in Table 2 demonstrate that ConfiGR delivers robust performance on both FB15K-237 and WN18RR. On each dataset, ConfiGR achieves competitive predictive accuracy—evidenced by comparable MRR and Hits@k scores—with state-of-the-art embedding-based methods. Notably, ConfiGR consistently outperforms vanilla DistMult across both benchmarks. Moreover, when compared with other HDC-based approaches, our method yields superior results, as indicated by the underlined scores in the table. Overall, these findings highlight that our hyperdimensional encoding enhances the quality of KG embeddings while providing a scalable and efficient solution for real-world knowledge graph representation.

Since we adopted the same GCN structure as CompGCN, our evaluation further confirms that the performance boost of our method originates primarily from the improved knowledge embeddings produced by our HDC encoder. Building on insights from [42], our approach demonstrates that even when employing an equivalent GCN aggregation strategy, the enhanced embeddings generated by our hyperdimensional encoding contribute significantly to superior link prediction performance. This indicates that the critical advantage of our method lies in the rich, context-aware representations it creates, rather than in additional graph structure aggregation.

6. Future work

While ConfiGR demonstrates strong performance across various graph-related tasks, several exciting directions remain for future exploration. One promising avenue involves extending the framework's evaluation to include larger-scale, real-world datasets in diverse application domains. For instance, recommender systems represent a natural fit for ConfiGR, where predicting user-item interactions in dynamic and sparse knowledge graphs could benefit from its scalability and robust encoding. Similarly, in biological networks, ConfiGR could facilitate tasks like drug discovery or modeling protein-protein interactions, where complex relationships require both accuracy and efficiency. In social networks, the framework's ability to process evolving relationships and detect community structures could be leveraged for tasks like influence maximization and information diffusion.

Another important area involves enhancing ConfiGR's robustness to noisy and sparse data, which are common challenges in real-world graphs. While preliminary results indicate the framework's hyperdimensional encoding is naturally resistant to noise, future studies could explore adaptive techniques to further optimize its performance in such settings. Comparisons to semantically meaningful datasets, such as those in the Open Graph Benchmark, would provide additional insights into its practical applicability.

Finally, while the current work focuses on undirected and unlabeled graphs, extending ConfiGR to handle directed labeled graphs more explicitly could unlock its potential for additional knowledge graph reasoning tasks. This includes addressing challenges associated with hierarchical or multi-relational data structures, such as temporal graphs or multi-modal networks. Investigating how ConfiGR can integrate with advanced machine learning approaches, such as hybrid models combining hyperdimensional computing with graph neural networks, could further enhance its versatility and expand its scope.

Through these extensions, ConfiGR has the potential to bridge the gap between theoretical advancements in hyperdimensional computing and practical applications in complex graph domains.

7. Conclusion

This study underlines the significance of graph analysis in representing real-world data, particularly emphasizing on the utilization of Hyperdimensional Computing (HDC) techniques for enhancing graph representations. It thoroughly examines the constraints associated with existing HDC-based graph representation methods and emphasizes the demand for innovative approaches that can adapt effectively to diverse graph types and tasks. To address this need, we introduce the ConfiGR: Configurable Graph Representation framework, which presents a promising avenue for harnessing the benefits of HDC while maintaining adaptability. To thoroughly assess its potential and performance, we conducted a comprehensive set of theoretical and empirical experiments, including general graph-related tasks and link prediction tasks utilizing knowledge graph datasets. Subsequently, we compared the results, highlighting the enhancements achieved in comparison to other established methods within the field.

CRediT authorship contribution statement

Ali Zakeri: Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Zhuowen Zou:** Conceptualization, Formal analysis, Investigation, Validation. **Hanning Chen:** Formal analysis, Investigation, Software, Validation. **Mohsen Imani:** Conceptualization, Funding acquisition, Project administration, Supervision, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mohsen Imani reports financial support was provided by Defense Advanced Research Projects Agency. Mohsen Imani reports financial support was provided by National Science Foundation. Mohsen Imani reports financial support was provided by Semiconductor Research Corp. Mohsen Imani reports financial support was provided by Office of Naval Research. Mohsen Imani reports financial support was provided by Air Force Office of Scientific Research.

Acknowledgements

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780, #2319198, #2321840 and #2312517, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

Appendix A

Theorem. *Assume n hypervectors of dimension D, creating the codebook $X = \{\boldsymbol{x}_i\}_{i\in[1,n]}$. The codebook is generated in a way that correlation between each two hypervectors comes from a Gaussian distribution: $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \sim \mathcal{N}(\mu, \sigma^2), 0 \leq \mu < 1$. With the similarity threshold denoted as ϵ, the memorization capacity m_c is calculated from the following equation:*

$$\epsilon = \Phi(-\frac{\mu m_c + 1 - \mu}{\sqrt{(\frac{1}{D} + \sigma^2)m_c - \sigma^2}})$$

where Φ is the cumulative distribution function (CDF) of the Gaussian distribution. In the case of random orthogonal codebook hypervectors ($\mu = 0, \sigma^2 = \frac{1}{D}$) the final value for m_c is calculated as below:

$$m_c = \frac{D}{2(\Phi^{-1}(\epsilon))^2} + \frac{1}{2}$$

Proof. We can derive the capacity of memorization in this codebook based on Equation (14) in the following fashion. We generate a memory hypervector $\boldsymbol{z}$ by bundling m hypervectors from X:

$$\boldsymbol{z} = \sum_{i=1}^{m} \boldsymbol{x}_i \rightarrow \boldsymbol{z} \in Span(x) \tag{A.1}$$

Assume $\boldsymbol{y}_0$ and $\boldsymbol{y}_1$ as two D-dimensional hypervectors, with the former being outside of the span of X, and the latter being a part of X. We want to compare the similarities of the memory with each of these hypervectors, which would detail the memorization power of our framework. Increasing the distance between these two distributions would lead to higher capacity and memorization power, with the framework being able to differentiate the two existing and non-existent patterns. The two distributions are derived as follows:

$$
\begin{aligned}
&\rightarrow \boldsymbol{y}_0 \notin Span(x):\\
&\qquad \langle \boldsymbol{z}, \boldsymbol{y}_0 \rangle = \sum_{i=1}^{m} \langle \boldsymbol{x}_i, \boldsymbol{y}_0 \rangle\\
&\Rightarrow \langle \boldsymbol{z}, \boldsymbol{y}_0 \rangle \sim \sum_{i=1}^{m} (\frac{2}{D}\mathcal{B}(D, 0.5) - 1)\\
&\qquad \sim \frac{2}{D}\sum_{i=1}^{m} \mathcal{B}(D, 0.5) - m\\
&\qquad \sim \frac{2}{D}\mathcal{B}(mD, 0.5) - m\\
&\qquad \rightarrow \begin{cases} \mathbf{E}[\langle \boldsymbol{z}, \boldsymbol{y}_0 \rangle] = 0 \\ \mathbf{Var}[\langle \boldsymbol{z}, \boldsymbol{y}_0 \rangle] = \frac{m}{D} \end{cases}
\end{aligned}
\tag{A.2}
$$

$$
\begin{aligned}
&\rightarrow \boldsymbol{y}_1 = \boldsymbol{x}_j \in Span(x):\\
&\qquad \langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle = \langle \boldsymbol{x}_j, \boldsymbol{y}_1 \rangle + \sum_{i=1, i\neq j}^{m} \langle \boldsymbol{x}_i, \boldsymbol{y}_1 \rangle\\
&\Rightarrow \langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle \sim 1 + \sum_{i=1, i\neq j}^{m} \mathcal{N}(\mu, \sigma^2)\\
&\qquad \sim 1 + \mathcal{N}((m-1)\mu, (m-1)\sigma^2)\\
&\qquad \rightarrow \begin{cases} \mathbf{E}[\langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle] = 1 + (m-1)\mu \\ \mathbf{Var}[\langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle] = (m-1)\sigma^2 \end{cases}
\end{aligned}
\tag{A.3}
$$

where $\mathcal{B}$ denotes the binomial distribution. In the case of random codebook generation we have:

$$
\begin{aligned}
&\rightarrow \mu = 0, \sigma^2 = \frac{1}{D}:\\
&\qquad \Rightarrow \begin{cases} \mathbf{E}[\langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle] = 1 \\ \mathbf{Var}[\langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle] = \frac{m-1}{D} \end{cases}
\end{aligned}
\tag{A.4}
$$

We measure the difference between the two distributions in this step, calculate the capacity. To do so, we can calculate the probability of the event that $\boldsymbol{y}_0$ is more similar to $\boldsymbol{z}$ than $\boldsymbol{y}_1$. The goal is to decrease this probability to reach a certain confidence level. The difference between the two distributions follows a Gaussian distribution with its mean as the difference of means and its variance as the sum of variances:

$$
\begin{aligned}
&\eta_0 = \langle \boldsymbol{z}, \boldsymbol{y}_0 \rangle \sim \mathcal{N}(0, \frac{m}{D})\\
&\eta_1 = \langle \boldsymbol{z}, \boldsymbol{y}_1 \rangle \sim \mathcal{N}(1 + (m-1)\mu, (m-1)\sigma^2)\\
&\Rightarrow \eta_1 - \eta_0 = \eta \sim \mathcal{N}(1 + (m-1)\mu, \frac{m}{D} + (m-1)\sigma^2)
\end{aligned}
\tag{A.5}
$$

The desired probability is calculated as below:

$$
\begin{aligned}
&\rightarrow Pr(\eta_0 > \eta_1) = Pr(\eta_1 - \eta_0 < 0)\\
&\qquad = Pr(\eta < 0) = \Phi(-\frac{1 + (m-1)\mu}{\sqrt{\frac{m}{D} + (m-1)\sigma^2}})
\end{aligned}
\tag{A.6}
$$

where Φ is the cumulative distribution function (CDF) of the Gaussian distribution. In the special case of a random codebook we have:

$$
\begin{aligned}
&\rightarrow \mu = 0, \sigma^2 = \frac{1}{D}:\\
&\qquad \Rightarrow Pr(\eta_0 > \eta_1) = \Phi(-\sqrt{\frac{D}{2m-1}})
\end{aligned}
\tag{A.7}
$$

Having derived the probability of mixing two different patterns, we can calculate the capacity m_c as the maximum number of hypervectors potentially memorized together, while being able to recover all of them. In the special case of random codebook hypervectors we have:

$$\begin{aligned} &\rightarrow Pr(\eta_0 > \eta_1) < \epsilon = \Phi(-\sqrt{\frac{D}{2m_c - 1}}) \\ &\Rightarrow m_c = \frac{D}{2(\Phi^{-1}(\epsilon))^2} + \frac{1}{2} \end{aligned} \tag{A.8}$$

where Φ^{-1} is the inverse distribution function for the standard Gaussian distribution. □

Theorem. *For the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$, $|\mathcal{E}| = e$, and the maximum degree is m, the sensitivity of the reconstructed node memory hypervectors for the inclusive scheme, exclusive scheme, and ConfiGR can be calculated using the following equations, respectively:*

$$\begin{aligned} S_1 &= \frac{1}{\sqrt{e}}\sqrt{D} \\ S_2 &= \frac{1}{\sqrt{n}}\sqrt{D} \\ S_3 &= \begin{cases} \frac{1}{\sqrt{n(\frac{m}{d})^d}}\sqrt{D}, & m > d \\ \frac{1}{\sqrt{n}}\sqrt{D}, & m \le d \end{cases} \end{aligned} \tag{A.9}$$

where d is the number of groups used in the ConfiGR representation.

Proof. Taking into account Equation (17), we can compute the sensitivity for distinguishing existing patterns from non-existing ones within the reconstructed node memory for each method. For the inclusive approach, we initiate the calculation by expanding Equation (2) to its complete form:

$$\begin{aligned} \hat{\boldsymbol{M}}_i &= \boldsymbol{G} \odot \boldsymbol{H}_i \\ &= 2\boldsymbol{H}_i \odot \sum_{(j,k)\in\mathcal{E}} \boldsymbol{H}_j \odot \boldsymbol{H}_k \\ &= 2\sum_{j\in Nbh(i)} \boldsymbol{H}_j + 2\sum_{j,k\neq i,(j,k)\in E} \boldsymbol{H}_i \odot \boldsymbol{H}_j \odot \boldsymbol{H}_k \end{aligned} \tag{A.10}$$

In its expanded form, the initial expression consists of two primary components. The first component represents the node memory hypervector, which encapsulates the existing patterns within $\hat{\boldsymbol{M}}_i$ as $\boldsymbol{x} \in H$. Simultaneously, the second component encompasses the crossterms, contributing solely to noise.

Considering two random hypervectors, $\boldsymbol{x}$ and $\boldsymbol{x'}$, both belonging to the space H such that $\langle \boldsymbol{x}, \boldsymbol{M}_i \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_i \rangle \approx 0$, the sensitivity for detecting $\boldsymbol{x}$ from $\boldsymbol{x'}$ within the reconstructed node memory can be derived as follows:

$$S_1 = \frac{\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] - \mathbf{E}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle]}{\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle]} \tag{A.11}$$

The initial expectation in the equation above can be expanded as below:

$$\begin{aligned} \mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] &= 2\sum_{j\in Nbh(i)} \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{H}_i \rangle] \\ &\quad + 2\sum_{j,k\neq i,(j,k)\in E} \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{H}_i \odot \boldsymbol{H}_j \odot \boldsymbol{H}_k \rangle] \end{aligned} \tag{A.12}$$

As $\boldsymbol{x}$ is drawn from the set H, the similarities in the second sum are approximately zero for random selections of $\boldsymbol{x}$, resulting in an expectation of zero. However, since $\boldsymbol{x}$ exhibits similarity with $\boldsymbol{M}_i$, at least one term within the first sum is non-zero. Furthermore, $\boldsymbol{x}$ can only be similar to a single vector in the orthogonal set of hypervectors H. Thus, we have: $\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] = 2$.

We arrive at a comparable expression for the second expectation in Equation (A.11). However, in this scenario, $\boldsymbol{x'}$ exhibits no similarity with $\boldsymbol{M}_i$ which implies that all the terms within the first sum are approximately zero. Likewise, the second sum evaluates to zero, mirroring the previous situation, as $\boldsymbol{H}_i \odot \boldsymbol{H}_j \odot \boldsymbol{H}_k$ consistently maintains orthogonality with any hypervector chosen from the set H. Therefore: $\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] = 0$.

To determine the last element in Equation (A.11), we begin by computing the standard deviation of the similarity between two orthogonal hypervectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, both of dimensionality D. Starting with the definition, we obtain:

$$\begin{aligned} \mathbf{std}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]^2 &= \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle^2] - \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]^2 \\ &= \mathbf{E}[\frac{1}{D^2}(\sum_{i=1}^{D} x_i y_i)^2] - 0 \end{aligned}$$

$$= \frac{1}{D^2}(\mathbf{E}[\sum_{i=1}^{D} x_i^2 y_i^2] + \mathbf{E}[\sum_{i,j=1,i\neq j}^{D} x_i y_i x_j y_j])$$
$$= \frac{1}{D^2}(D+0) = \frac{1}{D} \tag{A.13}$$

Returning to the main issue, we can calculate the standard deviation as follows:

$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle]^2 = 4 \sum_{j \in Nbh(H_i)} \mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{H}_i \rangle]^2 \tag{A.14}$$
$$+ 4 \sum_{j,k\neq i,(j,k)\in E} \mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{H}_i \odot \boldsymbol{H}_j \odot \boldsymbol{H}_k \rangle]^2$$

As previously mentioned, all similarity terms in both summations involve orthogonal hypervectors, and the number of terms is equal to the number of edges, denoted as e. This results in the following:

$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle] = \sqrt{(\frac{4e}{D})} = 2\sqrt{\frac{e}{D}} \tag{A.15}$$

which leads to the final sensitivity value for the inclusive method:

$$S_1 = \sqrt{\frac{D}{e}} \tag{A.16}$$

We can employ a similar proof to derive the sensitivity for the exclusive method. The reconstruction of the node memory hypervector for the model is as follows:

$$\hat{\boldsymbol{M}}_i = \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{H}_i)$$
$$= \rho^{-1}(\boldsymbol{H}_i \odot \sum_{j=1}^{n} \boldsymbol{H}_j \odot \rho(\prod_{k\in Nbh(j)} \boldsymbol{H}_k))$$
$$= \prod_{j\in Nbh(i)} \boldsymbol{H}_j \tag{A.17}$$
$$+ \sum_{j=1,j\neq i}^{n} \rho^{-1}(\boldsymbol{H}_i) \odot \rho^{-1}(\boldsymbol{H}_j) \odot \prod_{k\in Nbh(j)} \boldsymbol{H}_k$$

For this model, we should select the random hypervectors for existing and non-existent patterns differently: $\boldsymbol{x}$ and $\boldsymbol{x'}$ are chosen from $\prod_{j\in\mathcal{V}}^{m_i} \boldsymbol{H}_j$ such that $\langle \boldsymbol{x}, \boldsymbol{M}_i \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_i \rangle \approx 0$. Here, m_i represents the degree of node i, which also corresponds to the number of bound hypervectors in $\boldsymbol{M}_i$. With the expanded form, we can compute the necessary standard deviation and expectation in a manner similar to the previous model:

$$\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] = 1, \quad \mathbf{E}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle] = 0$$
$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle] = \sqrt{\frac{n}{D}} \tag{A.18}$$

It's important to note that, in this case, the number of terms summed together in $\hat{\boldsymbol{M}}_i$ is n, whereas it was e for the inclusive model. Consequently, the final result for the exclusive model is as below:

$$S_2 = \sqrt{\frac{D}{n}} \tag{A.19}$$

Lastly, for S_3, which represents the sensitivity measurement corresponding to ConfiGR, we have:

$$\hat{\boldsymbol{M}}_i = \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{H}_i)$$
$$= \rho^{-1}(\boldsymbol{H}_i \odot \sum_{j=1}^{n} \boldsymbol{H}_j \odot \rho(\prod_{k=1}^{d} \sum_{l\in Nbh(j), l\in s_k} \boldsymbol{H}_l))$$
$$= \prod_{j=1}^{d} \sum_{k\in Nbh(i), k\in s_j} \boldsymbol{H}_k \tag{A.20}$$
$$+ \sum_{j=1,j\neq i}^{n} \rho^{-1}(\boldsymbol{H}_j) \odot \rho^{-1}(\boldsymbol{H}_i) \odot \prod_{k=1}^{d} \sum_{l\in Nbh(j), l\in s_k} \boldsymbol{H}_l$$

$$= \sum \prod_{k=1, j \in s_k}^{d} \boldsymbol{H}_j + \sum_{j=1, j \neq i}^{n} \rho^{-1}(\boldsymbol{H}_j) \odot \rho^{-1}(\boldsymbol{H}_i) \odot (\sum \prod_{l=1, k \in s_l}^{d} \boldsymbol{H}_l)$$

In the last line, we transition from a product of sums to a sum of products over all possible selections. The choices for random hypervectors for existing and non-existent patterns in this case are as follows: $\boldsymbol{x}, \boldsymbol{x'} \in \{\prod_{j \in \mathcal{V}}^{d} \boldsymbol{H}_j\}$ such that $\langle \boldsymbol{x}, \boldsymbol{M}_i \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_i \rangle \approx 0$.

We begin the derivation of sensitivity in the case of ConfiGR:

$$\begin{aligned} \mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_i \rangle] &= \mathbf{E}[\langle \boldsymbol{x}, \sum \prod_{k=1, j \in s_k}^{d} \boldsymbol{H}_j \rangle] \\ &+ \sum_{j=1, j \neq i}^{n} \mathbf{E}[\langle \boldsymbol{x}, \rho^{-1}(\boldsymbol{H}_j \odot \boldsymbol{H}_i) \odot (\sum \prod_{l=1, k \in s_l}^{d} \boldsymbol{H}_l) \rangle] \\ &= 1 + 0 = 1 \end{aligned} \tag{A.21}$$

Keep in mind that $\boldsymbol{x}$ is similar to one of the constituents of the summation $\sum \prod_{k=1, j \in s_k}^{d} \boldsymbol{H}_j$ since it represents an existing pattern within the node memory hypervector. Furthermore, it's impossible for any of the terms in the subsequent summation to hold non-zero values, thanks to the binding of the hypervector $\rho^{-1}(\boldsymbol{H}_j \odot \boldsymbol{H}_i)$.

To compute the standard deviation, we also have:

$$\begin{aligned} \mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle]^2 &= \mathbf{std}[\langle \boldsymbol{x'}, \sum \prod_{k=1, j \in s_k}^{d} \boldsymbol{H}_j \rangle]^2 \\ &+ \sum_{j=1, j \neq i}^{n} \mathbf{std}[\langle \boldsymbol{x'}, \rho^{-1}(\boldsymbol{H}_j \odot \boldsymbol{H}_i) \odot (\sum \prod_{l=1, k \in s_l}^{d} \boldsymbol{H}_l) \rangle]^2 \end{aligned}$$

Determining the count of terms grouped together for standard deviation calculation is crucial. Assuming an average node degree in the graph is denoted as m, there are $(\frac{m}{d})^d$ terms bundled together in the summation $\sum \prod_{l=1, k \in s_l}^{d} \boldsymbol{H}_l$, resulting in a total term count of $n(\frac{m}{d})^d$. However, if m is smaller than the number of groups d, on average, meaning there's only one node in each group, then there will be a total of n terms. This results in:

$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_i \rangle]^2 = \begin{cases} \sqrt{\frac{n(\frac{m}{d})^d}{D}}, & m > d \\ \sqrt{\frac{n}{D}}, & m \leq d \end{cases} \tag{A.22}$$

Combining all the computations, we arrive at the outcome presented in Equation (18):

$$S_3 = \begin{cases} \frac{1}{\sqrt{n(\frac{m}{d})^d}} \sqrt{D}, & m > d \\ \frac{1}{\sqrt{n}} \sqrt{D}, & m \leq d \end{cases} \quad \square \tag{A.23}$$

Theorem. *For the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$, and the maximum degree is m, the bit operations required to decode the high-dimensional graph representation using the inclusive scheme, exclusive scheme, and ConfiGR can be calculated with the following equations, respectively:*

$$\begin{aligned} T_1 &= O(n^2 D) \\ T_2 &= O(n^2 D^2 (1 + \frac{m}{n} I_{n,m})) \\ T_3 &= O(n^2 D^2 (1 + \frac{d}{n} I_{n,d})) \end{aligned} \tag{A.24}$$

where d is the number of groups used in the ConfiGR representation, and $I_{n,k}$ is the number of iterations required for the resonator network to converge when decomposing k factors with a codebook size of n.

Proof. The decoding process in HDC's inclusive method is detailed in Section 3, where Equation (2) is employed to reconstruct the node memories. Subsequently, the similarity between these reconstructed memories and each node is measured to determine the existence of every possible edge.

In the node memory reconstruction process, there are a total of n operations, each involving calculations with D elements, resulting in a computation complexity of $O(nD)$. For edge reconstruction, it entails n^2 similarity checks between hypervectors of dimension D, leading to a complexity of $O(n^2 D)$. Therefore, the overall decoding complexity is given by: $T_1 = O(n^2 D)$.

Both of the subsequent methods also involve the two key steps: node memory reconstruction and edge reconstruction. The first step for the exclusive method is illustrated in Equation (5) and can be completed with a complexity of $O(nD)$. The exclusive model utilizes a resonator network in the next step to decompose the neighbors from the node memory hypervector, as demonstrated in Equation (6).

For each node memory, the algorithm initially computes the product of the codebook matrix with its transpose, denoted as HH^T. This holds $O(nD^2)$ bit computation, given that H possesses dimensions of $D \times n$. The resulting matrix is subsequently multiplied with the binding of m hypervectors during each iteration, an operation that can be completed with $O(mD + D^2) = O(D^2)$ bit computations. Overall, the algorithm is executed for m factors and runs for $I_{n,m}$ iterations, resulting in a complexity of $O(nD^2 + mD^2 I_{n,m})$ for each run. Since the decomposition process must be performed for all n nodes, this leads to an overall complexity of $T_2 = O(nD + n(nD^2 + mD^2 I_{n,m})) = O(n^2 D^2(1 + \frac{m}{n} I_{n,m}))$.

ConfiGR, similarly to previous approaches, initiates the decoding process by reconstructing the node memory hypervectors in $O(nD)$. The subsequent step is executed according to Equation (12). The computation of the product $H\bar{H}^T$ occurs once and takes $O(nD^2)$, while the process of determining a new guess for each factor is completed in $O(mD + D^2) = O(D^2)$. Through the utilization of groups, this algorithm can efficiently decompose memories with a smaller number of factors, significantly outpacing the exclusive method. This leads to $I_{n,d}$ iterations, where the number of factors is d. Considering this decomposition must be performed for all n nodes, the final complexity is calculated as $T_3 = O(nD + n(nD^2 + dD^2 I_{n,d})) = O(n^2 D^2(1 + \frac{d}{n} I_{n,d}))$. □

Data availability

Commonly-used public datasets have been used in this work.

References

[1] F. Chen, Y.-C. Wang, B. Wang, C.-C.J. Kuo, Graph representation learning: a survey, APSIPA Trans. Signal Inf. Process. 9 (2020) e15.
[2] S. Min, Z. Gao, J. Peng, L. Wang, K. Qin, B. Fang, Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks, Knowl.-Based Syst. 214 (2021) 106746.
[3] S. Kumar, A. Mallik, A. Khetarpal, B. Panda, Influence maximization in social networks using graph embedding and graph neural network, Inf. Sci. 607 (2022) 1617–1636.
[4] X.-M. Zhang, L. Liang, L. Liu, M.-J. Tang, Graph neural networks and their current applications in bioinformatics, Front. Genet. 12 (2021) 690049.
[5] M. Wang, L. Qiu, X. Wang, A survey on knowledge graph embeddings for link prediction, Symmetry 13 (2021) 485.
[6] C. Zhang, H. Yao, C. Huang, M. Jiang, Z. Li, N.V. Chawla, Few-shot knowledge graph completion, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 3041–3048.
[7] B. Shao, X. Li, G. Bian, A survey of research hotspots and frontier trends of recommendation systems from the perspective of knowledge graph, Expert Syst. Appl. 165 (2021) 113764.
[8] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, J. Leskovec, Qa-gnn: reasoning with language models and knowledge graphs for question answering, arXiv preprint arXiv:2104.06378, 2021.
[9] X. Huang, J. Zhang, D. Li, P. Li, Knowledge graph embedding based question answering, in: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 105–113.
[10] X. Tao, T. Pham, J. Zhang, J. Yong, W.P. Goh, W. Zhang, Y. Cai, Mining health knowledge graph for health risk prediction, World Wide Web 23 (2020) 2341–2362.
[11] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: a review of methods and applications, AI Open 1 (2020) 57–81.
[12] S. Ji, S. Pan, E. Cambria, P. Marttinen, S.Y. Philip, A survey on knowledge graphs: representation, acquisition, and applications, IEEE Trans. Neural Netw. Learn. Syst. 33 (2021) 494–514.
[13] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, T.-Y. Liu, Do transformers really perform badly for graph representation?, Adv. Neural Inf. Process. Syst. 34 (2021) 28877–28888.
[14] Y. Rong, W. Huang, T. Xu, J. Huang, Dropedge: towards deep graph convolutional networks on node classification, arXiv preprint arXiv:1907.10903, 2019.
[15] T. Zhao, X. Zhang, S. Wang, Graphsmote: imbalanced node classification on graphs with graph neural networks, in: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, 2021, pp. 833–841.
[16] A. Zakeri, Z. Zou, H. Chen, H. Latapie, M. Imani, Conjunctive block coding for hyperdimensional graph representation, Intell. Syst. Appl. 22 (2024) 200353.
[17] I. Nunes, M. Heddes, T. Givargis, A. Nicolau, A. Veidenbaum, Graphhd: efficient graph classification using hyperdimensional computing, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2022, pp. 1485–1490.
[18] D. Kleyko, D. Rachkovskij, E. Osipov, A. Rahimi, A survey on hyperdimensional computing aka vector symbolic architectures, part ii: applications, cognitive models, and challenges, ACM Comput. Surv. 55 (2023) 1–52.
[19] R.W. Gayler, S.D. Levy, A distributed basis for analogical mapping, in: New Frontiers in Analogy Research; Proc. of 2nd Intern. Analogy Conf., volume 9, 2009.
[20] R. Gayler, Multiplicative binding, representation operators and analogy, 1998.
[21] P. Kanerva, Binary spatter-coding of ordered k-tuples, in: International Conference on Artificial Neural Networks, Springer, 1996, pp. 869–873.
[22] T. Plate, et al., Holographic reduced representations: convolution algebra for compositional distributed representations, in: IJCAI, Citeseer, 1991, pp. 30–35.
[23] Y. Ma, M. Hildebrandt, V. Tresp, S. Baier, Holistic representations for memorization and inference, in: UAI, 2018, pp. 403–413.
[24] M. Nickel, L. Rosasco, T. Poggio, Holographic embeddings of knowledge graphs, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016.
[25] P. Poduval, A. Zakeri, F. Imani, H. Alimohamadi, M. Imani, Graphd: graph-based hyperdimensional memorization for brain-like cognitive learning, Front. Neurosci. (2022) 5.
[26] A. Zakeri, H. Chen, N. Srinivasa, H. Latapie, M. Imani, Enabling efficient and interpretable cybersecurity reasoning through hyperdimensional computing, IEEE Trans. Artif. Intell. (2025).
[27] E.P. Frady, S.J. Kent, B.A. Olshausen, F.T. Sommer, Resonator networks, 1: an efficient solution for factoring high-dimensional, distributed representations of data structures, Neural Comput. 32 (2020) 2311–2331.

[28] S.J. Kent, E.P. Frady, F.T. Sommer, B.A. Olshausen, Resonator networks, 2: factorization performance and capacity compared to optimization-based methods, Neural Comput. 32 (2020) 2332–2388.
[29] E.P. Frady, S. Sanborn, S.B. Shrestha, D.B.D. Rubin, G. Orchard, F.T. Sommer, M. Davies, Efficient neuromorphic signal processing with resonator neurons, J. Signal Process. Syst. 94 (2022) 917–927.
[30] A. Renner, L. Supic, A. Danielescu, G. Indiveri, B.A. Olshausen, Y. Sandamirskaya, F.T. Sommer, E.P. Frady, Neuromorphic visual scene understanding with resonator networks, arXiv preprint arXiv:2208.12880, 2022.
[31] E.P. Frady, D. Kleyko, F.T. Sommer, A theory of sequence indexing and working memory in recurrent neural networks, Neural Comput. 30 (2018) 1449–1513.
[32] A. Madkour, W.G. Aref, F.U. Rehman, M.A. Rahman, S. Basalamah, A survey of shortest-path algorithms, arXiv preprint arXiv:1705.02044, 2017.
[33] A. Kumar, S.S. Singh, K. Singh, B. Biswas, Link prediction techniques, applications, and performance: a survey, Phys. A, Stat. Mech. Appl. 553 (2020) 124289.
[34] S. Vashishth, S. Sanyal, V. Nitin, P. Talukdar, Composition-based multi-relational graph convolutional networks, arXiv preprint arXiv:1911.03082, 2019.
[35] B. Yang, W.-t. Yih, X. He, J. Gao, L. Deng, Embedding entities and relations for learning and inference in knowledge bases, arXiv preprint arXiv:1412.6575, 2014.
[36] Q. Li, Y. Zhong, Y. Qin, Mocokgc: momentum contrast entity encoding for knowledge graph completion, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 14940–14952.
[37] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, Adv. Neural Inf. Process. Syst. 26 (2013).
[38] M. Schlichtkrull, T.N. Kipf, P. Bloem, R.v.d. Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, Springer, 2018, pp. 593–607.
[39] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2d knowledge graph embeddings, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.
[40] S. Chen, X. Liu, J. Gao, J. Jiao, R. Zhang, Y. Ji, Hitter: hierarchical transformers for knowledge graph embeddings, arXiv preprint arXiv:2008.12813, 2020.
[41] Y. Chen, P. Minervini, S. Riedel, P. Stenetorp, Relation prediction as an auxiliary training objective for improving multi-relational graph representations, arXiv preprint arXiv:2110.02834, 2021.
[42] Z. Zhang, J. Wang, J. Ye, F. Wu, Rethinking graph convolutional networks in knowledge graph completion, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 798–807.