Intelligent Systems with Applications 22 (2024) 200353

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications

Conjunctive block coding for hyperdimensional graph representation

Ali Zakeri [a,*], Zhuowen Zou [a], Hanning Chen [a], Hugo Latapie [b], Mohsen Imani [a]

[a] University of California, Irvine, Irvine, CA, 92697, USA
[b] Cisco Systems, Inc., San Jose, CA, 95134, USA

ARTICLE INFO

Keywords:
Hyperdimensional computing
Vector symbolic architecture
Graph representation
Cognitive computing

ABSTRACT

Knowledge Graphs (KGs) have become a pivotal knowledge representation tool in machine learning, not only providing access to existing knowledge but also enabling the discovery of new knowledge through advanced applications. Among the scalable reasoning methods used for such applications, distributed graph embedding approaches, particularly GNNs, have become popular for large-scale graph-related tasks. However, many of these methods have limitations in their interpretability and fail to take into account structural similarity in their representation. Hyperdimensional Computing (HDC), also known as Vector Symbolic Architecture (VSA), addresses this issue by using well-defined cognitive operations on distributed representations of symbolic concepts. This work proposes and evaluates a new vector symbolic graph representation, CLOG, that preserves approximate structural similarity beyond edge correspondence and fundamentally differs from previous methods. The model's effectiveness in graph representation is evaluated through theoretical analysis, graph reconstruction experiments, and link prediction task, highlighting its efficiency and accuracy. This approach significantly advances the field by enhancing the capabilities of HDC in graph representation, representing a notable improvement over existing methods.

1. Introduction

Knowledge Graph (KG) as a type of knowledge representation has gained particular interest in the machine learning community. A knowledge graph is a multi-relational directed graph composed of entities as nodes and relations as edges. It collects and organizes relations and attributes about entities that play an increasingly important role in many applications, including question-answering and information retrieval. In addition to retrieving existing information from a knowledge graph, advanced applications such as link prediction and knowledge graph completion enable the discovery of novel and concealed knowledge within the graph (Chen et al., 2020).

Large-scale knowledge graphs like Freebase (Bollacker et al., 2008), WordNet (Fellbaum, 2010), and GeneOntology (Ashburner et al., 2000), are essential for AI applications including web/mobile search and QA. Traditional formal logic reasoning struggles with long-range reasoning in these extensive graphs (Zamini et al., 2022). Scalable methods like distributed graph embeddings are now favored for large-scale tasks, offering low-dimensional space representations for efficient graph algorithm execution. Graph neural networks (GNNs) are increasingly popular for their effectiveness in graph-related problems (Arora, 2020, Cheng et al., 2022, Park et al., 2019). However, GNNs often need ample labeled data for optimal performance, and knowledge graphs' long-tail nature, where many relations are found in few triples (Zhang et al., 2022), presents a challenge for purely data-driven methods.

Knowledge graph embedding approaches face two main limitations. Firstly, they transform symbolic, logical knowledge graphs into continuous representations without preserving entity and relation integrity, limiting gradient-based methods to post-hoc explanation (Hersche et al., 2023). Secondly, as highlighted by Zamini et al. (2022), many methods struggle with capturing multistep relationships and leveraging structural similarity for link prediction and graph completion. While Graph Neural Networks (GNNs) use network architecture to implicitly address these issues, the overlap between constituent (entity, relation) similarity and structural similarity often makes it unclear what drives predictions. Constituent similarity primarily arises from an element's intrinsic properties, whereas structural similarity is derived from its relationships with other elements.

To motivate the disentanglement of structural and constituent similarity, we turn to the study of analogical reasoning, a form of reasoning

* Corresponding author.
E-mail addresses: azakerij@uci.edu (A. Zakeri), zhuowez1@uci.edu (Z. Zou), hanningc@uci.edu (H. Chen), hlatapie@cisco.com (H. Latapie), m.imani@uci.edu (M. Imani).

https://doi.org/10.1016/j.iswa.2024.200353
Received 21 November 2023; Received in revised form 8 February 2024; Accepted 4 March 2024
Available online 8 March 2024

ubiquitous in human cognition that aims to identify a common relational system between two situations and infer from their commonalities (Gentner & Smith, 2013). As a method that relies heavily on the structural similarity between two sets of entities and their relations, it introduces as much insight as it introduces potential fallacy. Besides quoting a proverb to highlight patterns in common situations, people utilize analogical reasoning to draw connections between a known system to an unknown one to facilitate understanding and hypothesis generation (Gentner & Smith, 2013), while at times the analogy may generate factually incorrect inferences due to a superficial matching of entities and relations. By disambiguating the two types of similarity, we may provide better control of a model in its usage of them.

Hyperdimensional Computing (HDC), synonymously Vector Symbolic Architecture (VSA), addresses the issue of interpretability and analogical similarity through well-defined cognitive operations over distributed representation of symbolic concepts. HDC's design ensures the integrity of symbolic concepts within high-dimensional vectors (hypervectors), enabling interpretability. These representations allow recovery of graph nodes and edges as long as the graph's size is within the hypervector's memory capacity. Moreover, as the graph is represented by a single hypervector, the dot product of graph hypervectors reflects some global similarity between them. While various algorithms for graph representation exist (Gayler & Levy, 2009, Nunes et al., 2022, Poduval et al., 2022), they all produce graph hypervectors that determine structural similarity based on edge matching, facing scalability limits due to the hypervector's memory capacity.

This works aims to propose and evaluate a new vector symbolic graph representation that preserves approximate structural similarity beyond edge correspondence. We introduce CLOG: Conjunctive bLOck coding for hyperdimensional Graph representation, an approach that fundamentally differs from the previous methods. Our contributions are:

- A new hyperdimensional graph representation that leverages HDC's variable binding operator to preserve the conjunctive connections to neighbors for each node. A direct consequence is an increase in the memory capacity of the hypervectors for sparse graphs.
- A block encoding and masking scheme for projecting atomic concepts (graph nodes in this case) to hyperspace that leads to loose conjunction of the neighbors, enabling more efficient decoding of the representation.
- As CLOG is orthogonal to the approach GNN takes, we propose an integration of CLOG with GNN. To show CLOG's knowledge graph structure representation capability, we also test our model on benchmark link prediction datasets and compare it with existing approaches.

The following section covers the related literature in the field. Section 3 discusses the fundamentals of Hyperdimensional Computing and existing HDC graph representations. Section 4 presents the details to the proposed method and its components. Section 5 provides the experiment and evaluation of CLOG and its performance across various tasks.

2. Related work

2.1. Graph representation

Graph representation methods are a crucial area of research in computer science, particularly in the domains of data science, machine learning, and network analysis. These methods are designed to efficiently and effectively represent graph-structured data, which is inherently complex due to its non-Euclidean nature.

The pioneering work Perozzi et al. (2014) marked a foundational shift by applying natural language processing techniques to graph data, using truncated random walks to learn social representations. This concept was further refined in Grover and Leskovec (2016), which enhanced the model's flexibility to capture both local and global network structures. Building upon these embeddings, the domain of Graph Neural Networks (GNNs) emerged, with key contributions like Kipf and Welling (2016), extending deep learning to graph-structured data. The introduction of Graph Attention Networks (GATs) in Velickovic et al. (2017) added an attention-based mechanism, allowing for more nuanced node importance assessments. Further sophistication was seen in Dwivedi and Bresson (2020), extending the transformer architecture, originally designed for sequence data in natural language processing, to handle graph-structured data. Current research is focusing on enhancing the methods for greater scalability (Huang, Li, Cao, et al., 2022, Li et al., 2023, Rampášek et al., 2022, Thakoor et al., 2021), interpretability (Chen, Jiao, Liu, et al., 2022, Feng et al., 2022, Huang, Yamada, Tian, et al., 2022, Wu et al., 2023), and generalization, addressing real-world complexities in graph-structured data (Gao et al., 2023, Li, Zheng, Feng, et al., 2023, Li, Zhang, Cui, et al., 2023, Liu et al., 2023, Wang et al., 2022).

2.2. Hyperdimensional computing

The concepts underlying hyperdimensional computing have been established for quite some time, but it's only in recent years that they have begun to gain significant attention, both in theory and practice. Key theoretical developments (Clarkson et al., 2023, Frady et al., 2018) have drawn the attention of the broader machine learning community, leading to a wide spectrum of practical deployments, including health monitoring (Ge & Parhi, 2022, Moin et al., 2021, Ni, Lesica, Zeng, et al., 2022), reinforcement learning (Chen, Issa, Ni, et al., 2022, Ni et al., 2022), and lightweight recognition algorithms (Imani et al., 2022, Lee et al., 2023). This growing interest can be attributed to the way HDC capabilities address some of the shortcomings found in contemporary artificial neural networks, including higher learning efficiency (Hersche et al., 2022, Ni et al., 2024), better interpretability (Poduval et al., 2022), and natural parallelizability on hardware platforms (Chen et al., 2024). As a result, there's an emerging trend towards hybrid models that merge neuro-symbolic approaches (Hersche et al., 2023, Lee et al., 2023, Zou et al., 2022).

In particular, HDC has demonstrated substantial promise in competing with graph convolution neural networks (GCN) in the graph representation learning domain (Poduval et al., 2022). While HDC graph representation shares some similarities with the aggregation phase of GNNs in processing graph data, its approach and objectives differ significantly. GNNs are great for learning tasks but lack capabilities in high-level reasoning and knowledge extraction, as neural networks typically focus on learning rather than memorizing. In contrast, hyperdimensional learning uses symbolic hypervectors, mimicking robust brain-like reasoning and eliminating the need for iterative node aggregation across multiple layers (Chen et al., 2023, Kang et al., 2022, Nunes et al., 2022, Poduval et al., 2022).

3. Preliminaries

3.1. Hyperdimensional computing

Hyperdimensional Computing (HDC), synonymously Vector Symbolic Architecture (VSA) (Gayler, 1998, Kanerva, 1996, Plate et al., 1991), has the capability to manipulate data structures into distributed representations. HDC encodes data structures in high-dimensional vector spaces holographically, where random hypervectors, i.e., high-dimensional vectors, are used as bases for the representation and three main operations are employed to form an algebra over the high-dimensional space.

In this study, independent random bipolar vectors of dimension D are used as atomic primitives. The similarity between the vectors

are measured by their dot product, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \frac{1}{D} \sum_{i=1}^{D} x_i y_i$, which indicates whether or not the atomic objects are the same as well as the structural similarity between complex object. To construct structured objects, there are three HDC operations defined as below:

Bundling (+) operation is done via component-wise addition of hypervectors, denoted as $\boldsymbol{s} = \boldsymbol{x}_1 + \boldsymbol{x}_2$. This operation superposes the elements, acting as a memorization function that keeps the information of input data in the resulting vector. The bundled hypervector preserves similarity to its atomic hypervectors, i.e., $\langle \boldsymbol{s}, \boldsymbol{x}_1 \rangle \gg 0$; thus, it is well suited for representing sets. This similarity implies that even though information from multiple vectors is combined, the essence of individual vectors is not lost, facilitating associative memory retrieval within the HDC framework.

Binding (⊙) of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is done by the Hadamard product, i.e., component-wise multiplication, between the two hypervectors and denoted as $\boldsymbol{s} = \boldsymbol{x}_1 \odot \boldsymbol{x}_2$. The resulting conjunct hypervector $\boldsymbol{s}$ is dissimilar to its constituent vectors, $\langle \boldsymbol{s}, \boldsymbol{x}_1 \rangle \approx 0$. This operation is invertible, i.e., $\boldsymbol{x}_1 = \boldsymbol{s} \odot \boldsymbol{x}_2$. Since binding in some sense maintains the information of its constituents without increasing the size, it can function as variable association. This invertibility allows for the retrieval of original information, making binding a powerful tool for encoding complex relationships in HDC.

Permutation (ρ) operation, $\boldsymbol{s} = \rho(\boldsymbol{x})$, applies a cyclic shift on the components of $\boldsymbol{x}$. Permutation distributes over bundling, $\rho(\boldsymbol{x}_1 + \boldsymbol{x}_2) = \rho(\boldsymbol{x}_1) + \rho(\boldsymbol{x}_2)$, as well as binding, $\rho(\boldsymbol{x}_1 \odot \boldsymbol{x}_2) = \rho(\boldsymbol{x}_1) \odot \rho(\boldsymbol{x}_2)$. Like binding, permutation dissociates hypervectors, $\langle \rho^l(\boldsymbol{x}), \boldsymbol{x} \rangle \approx 0$ when $l \neq 0$. As permutation is reversible and induces a natural order via repeated permutation over the hypervector, it is used to represent sequences and encoding order for different levels in a hierarchy.

3.2. Graph representation in HDC

Despite some differences in algorithms and node ordering, existing HDC graph representations mainly leverage edge correspondence as measure of structural similarity (Gayler & Levy, 2009, Nunes et al., 2022, Poduval et al., 2022). For an undirected graph $G = (V, E)$, the framework first assigns a random bipolar hypervector $\boldsymbol{H}_i$ of size D to each node $i \in V$ in the graph, and generates the codebook matrix $H = [\boldsymbol{H}_1, \boldsymbol{H}_2, \cdots, \boldsymbol{H}_n]$ of the size $D \times n$ as the high-dimensional representation of graph nodes. As a result of random generation, the node hypervectors are nearly orthogonal, and the similarity between each pair of hypervectors is about zero: $\langle \boldsymbol{H}_i, \boldsymbol{H}_j \rangle \approx 0, i \neq j$.

For each node l, the HDC-based model creates a node memory hypervector by the set of its neighbors: $\boldsymbol{M}_l = \sum_{i \in Nbh(l)} \boldsymbol{H}_i$. Properties of the bundling operation implies that the similarity between the node memory and each of the bundled hypervectors is much greater than zero: $\forall i \in Nbh(l), \langle \boldsymbol{M}_l, \boldsymbol{H}_i \rangle \gg 0$. This property allows retrieval of each neighbors during reconstruction process through similarity check.

The model then constructs a single hypervector for the full graph in two steps. First, it associates (binds) each node hypervector with the corresponding node memory, $\boldsymbol{H}_i \odot \boldsymbol{M}_i$. This adds a distinction between the node memories and allows the model to recognize the correct memory for each node during the graph reconstruction. Then, all the associated pairs are bundled together into the graph hypervector $\boldsymbol{G}$:

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

The last equality uses the distributivity of binding over bundling. The result is a transparent model that has compressed all the edges in the graph, where each edge is represented by the binding of endpoint hypervectors. It can be used to reconstruct the original graph or perform reasoning tasks on it.

To retrieve the graph information from the graph hypervector, the HDC-based representation framework first recovers each node's memory, which can be approximately done by binding the graph hypervector $\boldsymbol{G}$ with the corresponding node hypervector:

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

The obtained node memory is denoted as $\hat{\boldsymbol{M}}_l$ and differs from the original node memory hypervector $\boldsymbol{M}_l$ by including extra components. In the subsequent and concluding phase of the decoding process, these excess components are eliminated, thanks to the orthogonality of hypervectors.

Having estimated the node memory, we can inspect the connection between nodes i and l by measuring the similarity between the node hypervector of one and the node memory hypervector of the other, $\langle \boldsymbol{H}_i, \hat{\boldsymbol{M}}_l \rangle$. If there exists an edge between i and l, then $\langle \boldsymbol{H}_i, \hat{\boldsymbol{M}}_l \rangle \gg 0$. Otherwise, $\langle \boldsymbol{H}_i, \hat{\boldsymbol{M}}_l \rangle \approx 0$.

4. Main contribution

This section covers the method we propose to represent the structure of the graph in high-dimensional space. Hyperdimensional mathematics, as outlined in Section 3, is employed to distribute the graph information across a fully holistic high-dimensional representation. We introduce CLOG, which employs a unique two-stage block encoding and an innovative decoding algorithm. The details of this design are discussed in depth in this section.

4.1. Resonator network

As discussed in Section 3.2, previous HDC-based work on graph representation mainly utilize the bundling operation and its characteristics to aggregate the node connections into memory hypervectors. Using bundling to memorize hypervectors restricts the scalability of the model, as the resulting memory hypervectors will be much larger in magnitude than their atomic constituents, which leads to eventual saturation of the memory. This saturation can be observed as the decoding process becomes more challenging, and the number of noise terms increases while attempting to determine the similarity between the memory and its constituents.

To overcome this, memorization can be performed by binding hypervectors, eliminating the mentioned drawback. However, this approach can be challenging when it comes to decoding the information from the conjunct hypervectors.

While the binding operation is reversible once all but one of the constituents is known, finding all constituent vectors from a given bound hypervector is not trivial. In general, this factorization problem may involve two or more factors, leading to exponential growth in the potential solutions space. Assume we are given a composite vector $\boldsymbol{s}$, formed as a product of m vectors:

$$\boldsymbol{s} = \boldsymbol{s}_1 \odot \boldsymbol{s}_2 \odot \cdots \odot \boldsymbol{s}_m \tag{3}$$

where the factors $\boldsymbol{s}_i$ are drawn from codebooks $X_i = \{\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{in}\}$. The task in the factorization problem is to find the factors $\boldsymbol{s}_i$, given the composite vector $\boldsymbol{s}$ and codebooks X_i.

The resonator network proposed by Frady et al. (2020), Kent et al. (2020), and further utilized in various applications (Frady et al., 2022, Renner et al., 2022), attempts to solve this problem without exhaustively searching through all possible combinations of the factor, as shown in Fig. 1. Initializing a guess for each factor by using superposition on all the vectors in each corresponding codebook, the algorithm iteratively improves the guesses and infers new estimates from the ones in the last iteration.

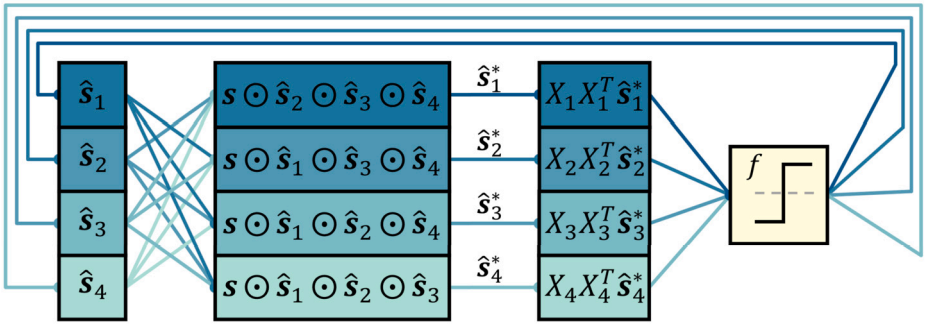

Fig. 1. Structure of the resonator network, with the number of factors set to 4. The algorithm progressively refines its guesses for the constituents of s, $\hat{s}_1$ to $\hat{s}_4$, deriving new estimates based on those from the previous iteration.

The vector $\hat{s_i}$ represents the current estimate for each factor. First, the estimates are initialized to be all possible factors bundled together, i.e., $\hat{s}_i(0) = \sum_{j=1}^{n} x_{ij}$. A particular factor can then be inferred from s based on the estimates for the other ones:

$$\hat{s}_i^*(1) = s \odot \hat{s}_1(0) \odot \cdots \odot \hat{s}_{i-1}(0) \odot \hat{s}_{i+1}(0) \odot \cdots \odot \hat{s}_m(0) = s \odot \prod_{j \neq i}^{m} \hat{s}_j(0) \tag{4}$$

where $\prod$ is the notation used for binding multiple vectors. Since binding distributes over bundling, the vector product $\hat{s}_1(0) \odot \hat{s}_2(0) \odot \cdots \odot \hat{s}_{i-1}(0) \odot \hat{s}_{i+1}(0) \odot \cdots \odot \hat{s}_m(0)$ expresses every combination of factors (with a total of n^{m-1}) during the first iteration, allowing many potential combinations of the factors to be considered at once when deducing the ith factor.

However, the procedure for inferring new factors is noisy if many guesses are processed simultaneously. This issue is resolved by projecting the noisy estimate $\hat{s}_i^*$ back to the span of original codebooks, that contain all the possible factors of the input s. Formally, combining the mentioned processes leads to the following update equation:

$$\hat{s}_i(t+1) = f(X_i X_i^T \hat{s}_i^*(t+1)) \tag{5}$$

where the function f prevents the diverging positive feedback by thresholding the vector elements to ± 1, and combines with the matrix multiplication of $X_i X_i^T$ to transform the noisy guess $\hat{s}_i^*(t+1) = \hat{s} \odot \hat{s}_1(t) \odot \ldots \odot \hat{s}_{i-1}(t) \odot \hat{s}_{i+1}(t) \odot \ldots \odot \hat{s}_m(t)$ to the span of the codebook. $X_i^T \hat{s}_i^*(t+1)$ computes the similarity of the noisy guess with the codebook, which acts as a measure of confidence for the possibility of each codebook vector being a factor in s. The final multiplication with X_i then applies the projection to the desired span of possible factors, using the confidence levels as weights.

By repeatedly executing the specified operations, the resonator network can generate improved estimates while decreasing the level of noise in the guesses, until the model reaches convergence and the solution is discovered.

4.2. Block encoding & masking

Even though the resonator network is a great and efficient approach to factorizing composite hypervectors, it suffers from scalability issues. Fig. 2 demonstrates changes in the accuracy and iterations of the method with changes in the size of its search space, i.e., the number of possible factor combinations. It is evident that the accuracy of the network can be maintained only in a very small range of the search space size, which limits the size and density of the input graphs in our case as we utilize the method later in our design. It is also apparent that trials on larger graphs lead to the algorithm's inability to reach convergence, resulting in a decrease in accuracy. Further increasing the search space would lead to spurious points and low final accuracy as well.

To deal with the scalability issues, we propose a configuration in the base encoding method that allows the resonator network to work with smoother search spaces, enabling more efficient decoding of the representation. We also change the update rule for the resonator network accordingly, which significantly decreases the number of iterations needed for the network to converge.

Instead of generating a random hypervector for each node, we split each node hypervector into a number of blocks, only randomly generating a specific portion of them, leaving other blocks as vectors with all the components set to 1. We define density, η, as the proportion of random blocks to the total number of blocks:

$$\eta \in \{\frac{1}{d}, \frac{2}{d}, \frac{3}{d}, ..., 1\}, \quad \text{where } d = \text{Number of blocks} \tag{6}$$

This practice can also be represented by applying a 1-mask, δ, on the hypervector:

$$(x[\delta])_i = \begin{cases} x_i & \text{if } \delta_i = 0 \\ 1 & \text{if } \delta_i = 1 \end{cases} \tag{7}$$

An example of such masking can be seen in Fig. 3(a).

Now suppose we are working on a set of hypervectors with masked blocks in a factorization problem. The resonator network is able to update its guesses more smoothly throughout its search space, as the previously orthogonal hypervectors now have correlations between them. Additionally, it can be seen that by adding unit blocks in between the randomly generated ones, each block in the final composite hypervector would have less number of factors involved in its creation, an example of which is provided in Fig. 3(b). We could exploit this new characteristic of the conjunct vectors by changing the update rule of the resonator network and including new termination rules, hence increasing its speed of convergence.

We propose the new update rule for the resonator network as below:

$$\hat{s}_i(t+1) = f(\frac{1}{\eta} X_i[\Delta] X_i[\bar{\Delta}]^T \hat{s}_i^*(t+1)) \tag{8}$$

where Δ is the matrix of 1-masks, i.e., concatenation of the masks applied to each hypervector, and $\bar{\Delta}$ contains the corresponding 0-masks. In other words, $X_i[\bar{\Delta}]$ is the same as $X_i[\Delta]$, with the **1** blocks replaced by **0** blocks. This modification specifically negates the effect of excessive correlation between hypervectors, which is a result of the introduction of **1** blocks. Also, multiplication with $\frac{1}{\eta}$ is needed for the network to maintain the magnitude of the guesses.

The resonator network terminates its iterations when all factors have reached convergence, meaning that further iteration does not lead to any changes in the guesses. The same criterion works for the design we propose as well, but the use of masks and block hypervectors can be significantly advantageous here. By splitting the hypervector into blocks and updating the resonator network accordingly, new criteria can be added to ensure the convergence of individual blocks.

Suppose the resonator network is on its k^{th} iteration, currently holding onto the guesses $\hat{s}_i$ for the factors building the composite hypervec-

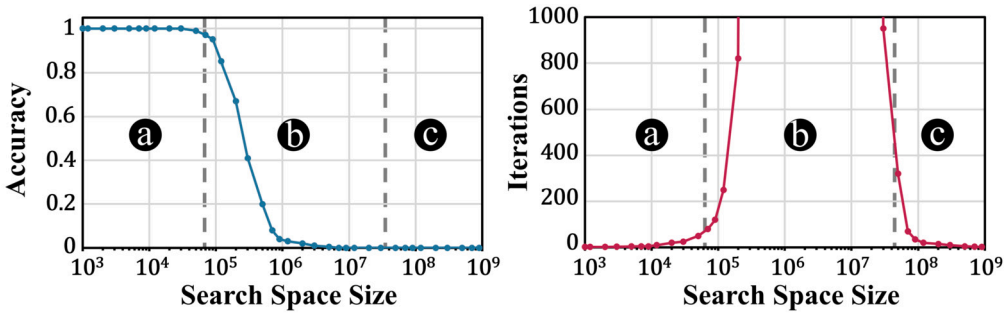

Fig. 2. Convergence behavior of the resonator network. The plots can be split into three sections: (a) is where the algorithm converges to the correct point in its search space, (b) is where the factorization fails to converge and the performance deteriorates as the search space size grows, and (c) is the region where the resonator network converges to spurious points.

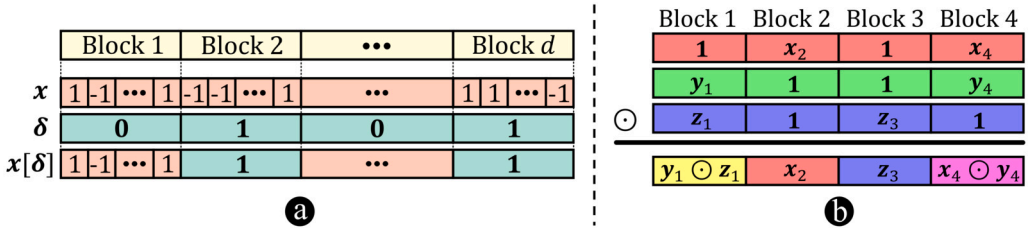

Fig. 3. (a) Example of a mask being applied to blocks of a randomly generated hypervector. $\boldsymbol{\delta}$, with a density of $\frac{2}{d}$, masks the blocks 2 and d of the hypervector $\boldsymbol{x}$ with blocks of $\mathbf{1}$. (b) Example of binding three masked hypervectors. It is evident that each block might contain less number of bound blocks than the number of hypervectors.

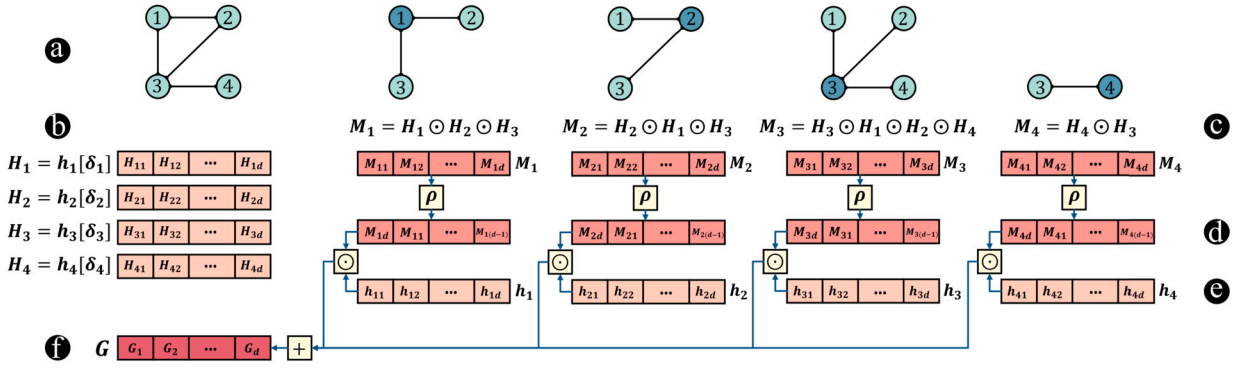

Fig. 4. Overview of the CLOG encoding design. (a) The sample graph consists of 4 nodes and 4 edges. (b) The node hypervectors $\boldsymbol{h}_i$ are generated and also converted to a masked version $\boldsymbol{H}_i$. (c) The node memory hypervectors are generated as described by Equation (9). (d) A block permutation is performed on the node memories. (e) The results are binded with the full node hypervectors. (f) The resulting calculations are bundled into the final graph hypervector $\boldsymbol{G}$.

tor $\boldsymbol{s}$. We can compare $\boldsymbol{s}$ with the hypervector built from binding all of the current guesses, denoted as $\hat{\boldsymbol{s}}$. If the similarity between the two mentioned vectors is high, i.e., $\langle \boldsymbol{s}, \hat{\boldsymbol{s}} \rangle \gg 0$, we can conclude that the guesses have converged to their respective codebook entries. Assuming that the hypervector was split into blocks as previously described, the mentioned criteria can be checked for each block, independent of the others. Also taking masking into consideration, it can be seen that some of the blocks might contain a number of masked blocks as their factors. If 1-masks are used for such cases, they can be disregarded as a factor, since they act as the identity element in binding, implying that individual blocks might have been built using fewer factors than the full hypervector. This can significantly speed up the process of detecting convergence; if the convergence criteria are satisfied for individual hypervector blocks, they can be inspected and hint at factors for the full hypervector.

4.3. CLOG: HDC-based graph representation

We now propose our novel graph memorization and learning scheme, CLOG. We break the design in two parts, first explaining the encoding process in detail, and then moving on to the decoding procedure. Fig. 4 presents a high-level view of the design.

4.3.1. Encoding

In the new approach CLOG, we use a two-step encoding that reduces the number of terms bundled together, allowing the model to scale better with the size and density of the graph. This introduces an increase in the memory capacity of hypervectors, addressing scalability issues present in previous methods. To start, we generate a random hypervector of dimension D, $\boldsymbol{h}_i$, for each node. Using a fixed number of blocks d, we also apply a random mask $\boldsymbol{\delta}_i$ of density η to each node hypervector, resulting in $\boldsymbol{H}_i$.

Having generated the necessary vectors to represent each node, the graph encoding can now be separated into two levels of memorization. For the first level, CLOG captures the information of nodes locally, associating all the neighbors for each node together in a single node memory hypervector $\boldsymbol{M}_l$:

$$\boldsymbol{M}_l = \boldsymbol{B}_l \odot \prod_{i \in Nbh(l)} \boldsymbol{H}_i \tag{9}$$

where $\boldsymbol{H}_i$ is the masked version of the full node hypervector, i.e., $\boldsymbol{H}_i = \boldsymbol{h}_i[\boldsymbol{\delta}_i]$, and $\boldsymbol{B}_l$ is the base node memory, which is also a masked hypervector. (Note that $\boldsymbol{M}_l = \boldsymbol{B}_l$ if the node doesn't have any neighbors.) To avoid generating more hypervectors for each node, we can assign the masked node hypervector as its respective base node memory, $\boldsymbol{B}_l = \boldsymbol{H}_l$, assuming that the graph doesn't contain any self-loop. Leveraging HDC's variable binding operator in the first step allows CLOG to preserve the conjunctive connections to neighbors for each node, exhibiting a fundamental shift from previous methods that mostly focused on edge correspondence.

For the second encoding level, all the information gathered in the node memories is bundled together in the final graph hypervector:

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

where $\boldsymbol{h}_i$ is the full node hypervector, and a permutation is applied to node memories as well to completely separate the two levels of encoding. It is worth mentioning that the amount of permutation is equal to the length of hypervector blocks, which ensures that the contents of different blocks are not mixed together.

4.3.2. Decoding

The reconstruction process can be divided into two steps. First, an estimate of the node memory for each node is retrieved from the graph hypervector:

$$\hat{\boldsymbol{M}}_i = \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{h}_i) \tag{11}$$

Note that we use the length of hypervector blocks as the size of permutation here as well, essentially shifting the blocks back by one. The result of the computation is denoted $\hat{\boldsymbol{M}}_i$, which is different than the original node memories and may contain noise terms. For the next step, CLOG needs to identify each node's neighbors from its node memory hypervector. Since the model creates node memories through binding, the resulting bound hypervectors can be factorized into their constituents using the resonator network, which provides an efficient mechanism for hypervector factorization, as discussed in Section 4.1. We also apply the modifications discussed in Section 4.2 to the resonator network, following the use of block encoding and masking in the encoding step, to add to the efficiency and scalability of the approach. Here we will use the following equation:

$$\hat{\boldsymbol{H}}_i(t+1) = f\left(\frac{1}{\eta} H \bar{H}^T \left(\hat{\boldsymbol{M}}_l \odot \boldsymbol{B}_l \odot \prod_{j=1, j\neq i}^{m} \hat{\boldsymbol{H}}_j(t)\right)\right) \tag{12}$$

which associates all previous guesses for factors with the base vector and the composite vector (i.e., the retrieved node memory), and then computes a weighted linear combination of codebook entries as the new guess for the current factor. Also note that the number of factors used to build each node memory is assumed to be m, the maximum node degree.

The resonator network would update its guesses for the factors, iterating from the first factor to the last one, and then going back to the first one for a new set of guesses. The iterations will end when all factors have reached a stable point, where the new guesses for each factor remain unchanged from the previous iteration. This process needs to be performed for each node for CLOG to acquire the neighbors for all the nodes, hence reconstructing the full graph.

With the use of block encoding and masking, we are able to modify the algorithm to find the factors much earlier than the convergence of the full hypervector, while also working with a smaller number of factors than m. Since the blocks are essentially independent of each other, we perform a block-wise similarity check every few iterations. If there are blocks where the binding of guesses matches the conjunct vector, it implies that those blocks are converged. Beginning with a single factor, i.e., $m = 1$, the algorithm increases the number of factors while identifying blocks that could be constructed with fewer factors, resulting in faster convergence. This is a direct result of the masking approach; some of the blocks might contain more than one masked factors, which helps the resonator network to identify and factorize them with a smaller number of factors and search space size.

CLOG introduces significant advancements in hyperdimensional computing for graph representation compared to previous models. Unlike earlier models focused primarily on edge correspondence, this framework uses the binding operation as the mean for connecting neighboring nodes with each other, and incorporates a unique block encoding and masking scheme, ensuring a looser conjunction of neighbors, which enhances the decoding performed through our modified resonator network. This approach not only preserves structural similarity beyond simple edge connections but also improving memory capacity and efficiency in graph representations compared the previous frameworks.

5. Experiments

In this section, we will display the experiments carried out on CLOG and discuss their significance. We conduct three types of experiment on our model. First, we theoretically analyze the mathematical properties of CLOG and display its competence by elaborating on the characteristics of our design. Second, we demonstrate the effectiveness of CLOG in reconstructing graphs from encoded graph hypervectors. We assess our model empirically by measuring several performance metrics when faced with the reconstruction task in a variety of input and model hyperparameter settings. Lastly, we examine the model's capabilities on the link prediction task for knowledge graphs. We evaluate our results using benchmark datasets and compare them to several recent studies on the task.

5.1. Theoretical evaluation

Having explained the underlying principles of CLOG, we will now compare it to prior research on HDC-based graph representation and point out the main distinctions through a theoretical examination. We use GrapHD (Poduval et al., 2022) from the previous works, and also include an intermediate model in the comparisons, which uses the same approach as our design but does not utilize the masking and block encoding procedures. Previous graph representation, CLOG without block encoding, and CLOG with block encoding are indexed $1, 2, 3$ respectively. Fig. 5 illustrates a comparison between two major aspects of the models, computational complexity and memory sensitivity.

It is crucial to compare the amount of computation needed for each model to perform its reconstruction. All the models use the same order of computation for encoding the graph representation, but the decoding process varies in complexity among the three models, which can be quantified as follows:

Theorem 1. *Given the graph $G = (V, E)$, with size $|V| = n$ and maximum degree m, the time complexity of decoding the high-dimensional representation of the graph using each of the three models can be quantified as below:*

$$\begin{aligned} T_1 &= O(n^2 D) \\ T_2 &= O(n^2 D^2 (1 + \frac{m}{n} L_{n,m})) \\ T_3 &= O(n^2 D^2 (1 + \frac{1}{n} \sum_{k=0}^{m} \binom{m}{k} (1-\eta)^k \eta^{(m-k)} k L_{n,k})) \end{aligned} \tag{13}$$

where $L_{n,k}$ is the number of iterations needed for the resonator network to converge when decomposing k factors with a codebook size of n.

Proof. See proof in Appendix A □

Assuming that the hyperparameters of the model are fixed, Fig. 5(a) illustrates the computational complexity measurements, calculated using Equation (13). The values for CLOG exhibit a near-quadratic in-

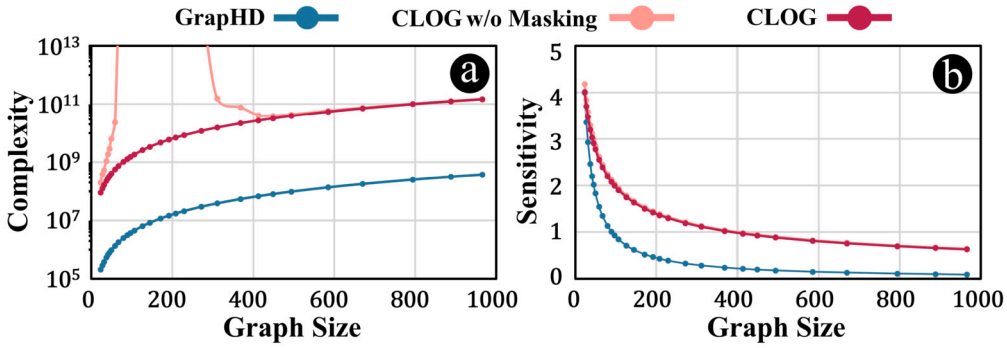

Fig. 5. Comparison between GrapHD and CLOG without and with masking. (a) Computation complexity and (b) Sensitivity for decoding the reconstructed node memory as a function of graph size. The values for $L_{n,m}$ for CLOG w/o masking can be seen in Fig. 2, while $L_{n,k}$ is seen to be approximately similar across different values of k, provided that an appropriate block density is used (the ablation study in Section 5.2 shows that $\eta \in [0.3, 0.5]$ is the best range of values for density.) Hypervector dimensionality and graph density are fixed: $D = 400$, density $= 0.1$.

crease with respect to the graph size used for reconstruction, under set values for graph density. The intermediate model follows a similar trend, before reaching a point where the complexity infinitely grows as we increase the graph size, with the model completely failing to perform the task. The model is able to perform the reconstruction but achieves spurious results for bigger graphs. Lastly, complexity of GrapHD increases quadratically under similar conditions, and its values are smaller than CLOG by a factor of dimensionality.

Another principal attribute of a representation method is the quality of the result it produces. In the case of HDC-based models, we can evaluate the precision of the hypervectors used for representation. We can quantify the quality of stored vectors in the memory hypervectors by the sensitivity of the signal relative to noise, formally defined as follows:

Definition *(Sensitivity).* Sensitivity for detecting an existing pattern $\boldsymbol{x}$ from a non-existent pattern $\boldsymbol{x'}$ in the hypervector $\boldsymbol{z}$ can be mathematically defined as below:

$$S := \frac{\mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{z} \rangle] - \mathbf{E}[\langle \boldsymbol{x'}, \boldsymbol{z} \rangle]}{\mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{z} \rangle]} \tag{14}$$

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

When storing through bundling, the sensitivity for detection of each constituent in the memory hypervector decreases as more hypervectors are added. More detailed explanation of the definition above can be found in Frady et al. (2018). In order to quantify the power of graph representation for each approach, we can calculate the sensitivity values for the reconstructed memories, which helps us demonstrate the difference in memorization power and capacity of the methods.

Theorem 2. *Given the graph $G = (V, E)$, with cardinalities $|V| = n, |E| = e$, and maximum degree m, sensitivity of reconstructed node memory hypervectors can be determined for each of the three methods using the following formulas:*

$$\begin{aligned} S_1 &= \frac{1}{\sqrt{e}}\sqrt{D} \\ S_2 &= \frac{1}{\sqrt{n}}\sqrt{D} \\ S_3 &= \frac{1 - \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}{\sqrt{n - \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}}\sqrt{D} \\ \textit{where} \quad p_{k,m} &= \frac{\binom{m+1}{m+1-k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}} \end{aligned} \tag{15}$$

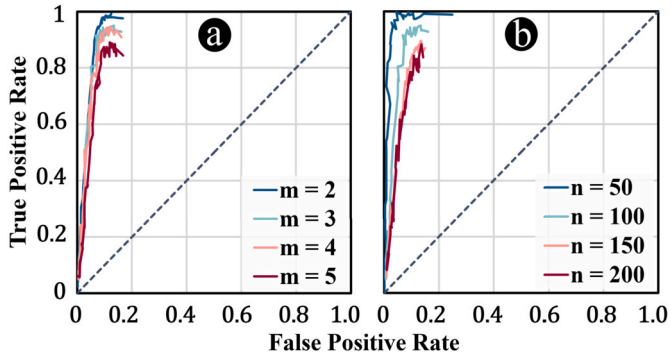

Fig. 6. ROC curves for CLOG, under different sets of parameters. (a) The graph size and average degree are fixed: $n = 100, \bar{m} = 2$, with the maximum degree of the graph m changing across four values (b) The maximum and average degree of the graph are fixed: $m = 3, \bar{m} = 2$, with the graph size changing across four values.

Fig. 5(b) shows a plot of the sensitivity values of each method for different graph sizes. It can be seen that the GrapHD holds the lowest sensitivity evaluation, CLOG comes in second, and CLOG without masking has the highest sensitivity with slightest of margins over CLOG. Note that the graph density is set to a fixed value across the measurements, leading to the following relation between the number of nodes and edges in the graph: $e = O(n^2)$.

Combining the two evaluations shown in Fig. 5, it can be inferred that CLOG is the leading algorithm between the three, since it has a clear advantage in sensitivity over previous work with the cost of a slight increase in complexity. CLOG also surpasses the intermediate design without masking, while maintaining a high level of memorization quality. Not utilizing masking and block encoding leads to infinite growth in complexity for larger graphs and eventually failing to produce acceptable results for the approach.

Considering two major properties of the frameworks, memory sensitivity and computational complexity, the theoretical evaluation presents the improvements CLOG brings to the table. Memory sensitivity, a key attribute of a representation method, evaluates the precision of the hypervectors used for representation. This is quantified by the sensitivity of the signal relative to noise. CLOG is demonstrated to have high sensitivity with a clear advantage in sensitivity over previous work, indicating superior memorization power and capacity. Computational complexity is crucial in determining the amount of computation needed for each model to perform its reconstruction, in which CLOG manages the trade-off between complexity and quality effectively, maintaining a reasonable amount of computation compared to previous frameworks.

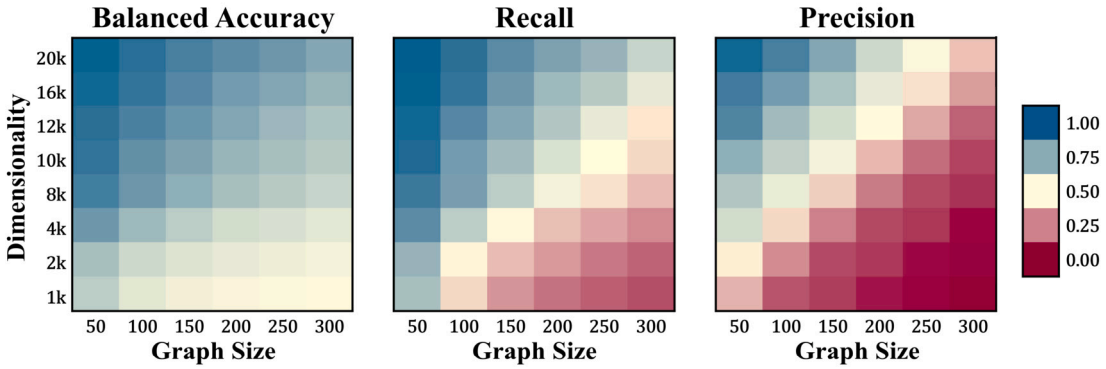

Fig. 7. Classification accuracy plot for CLOG. The heatmaps illustrate the values for three scoring metrics of balanced accuracy, recall, and precision. The reconstruction task is performed for graphs of varying sizes with fixed maximum and average degree values $m = 4, \bar{m} = 2$, using a range of dimensionalities for the model.

The theoretical assessment of CLOG showcases the improvements of the framework over previous work by focusing on two crucial aspects: memory sensitivity and computational complexity. Memory sensitivity measures the accuracy of the hypervectors in representation, with CLOG showing enhanced sensitivity and superior memory capability compared to previous methods. Meanwhile, computational complexity, which gauges the necessary computational effort for model reconstruction, is effectively balanced by CLOG. It avoids the pitfalls of excessive computational demand, ensuring it remains at a similar order compared to earlier frameworks.

In addition, the adjustments provided by masking and block encoding prove to be greatly beneficial, which is demonstrated in the comparison with the intermediate method without masking. The absence of these modifications leads to infinite growth in complexity for larger graphs, whereas CLOG handles this effectively, without compromising memorization quality

5.2. Graph reconstruction experiments

We also conducted numerous experiments to evaluate our model in addition to the theoretical assessments mentioned. CLOG can be seen as a classifier when tackling the graph reconstruction task. Each graph includes a number of edges, that can be given the label “1” or “existing edge”, while the missing edges when compared to the complete graph of the same size can be labeled as “0” or “non-existent edge”. Thus, the objective of the graph reconstruction task can be stated as classifying the edges of the graph. This allows us to evaluate CLOG using classification performance measurements.

We first illustrate and discuss the receiver operating characteristic (ROC) curve for our design, which shows the diagnostic ability of the classifier as the decision threshold is changed. Fig. 6 demonstrates a series of ROC curves generated from the performance of the model on different types of graphs. We can observe that the curves don’t necessarily start at $(0, 0)$ and end at $(1, 1)$ coordinates, as ROC curves generally do, as the decision threshold is used in CLOG unlike most classifiers; it affects multiple stages of the classification procedure, (such as checking the hypervector similarities during the factorization of node memories) rather than solely the final stage. Regardless of this difference, the curves demonstrate the capability of our model for various groups of graphs, being close to the upper left corner, i.e., perfect classification, in the ROC space. As shown in Fig. 6(a), the performance of our model decreases slightly as the maximum degree increases, yet it still maintains a satisfactory level of edge classification accuracy. Fig. 6(b) also illustrates the impact of graph size on the classification performance of our model, with the degree parameters fixed. Although the curves decline as the graph size increases, the model still performs acceptably well.

Fig. 7 shows the classification performance metrics for CLOG, when conducting the reconstruction task at various dimensionalities and graph sizes. Note that balanced accuracy is utilized instead of accuracy as an overall performance metric for the model, as accuracy can be a misleading metric for imbalanced datasets, such as sparse graphs. The experiments were done on graphs generated with fixed values for average and maximum degree, and the encoding hyperparameters also kept unchanged. The results suggest the effectiveness of our model.

It is of significant importance to analyze the behavior of our model with changes in its hyperparameters as well. CLOG employs three main parameters for working with hyperdimensional vectors: dimensionality, number of blocks, and density. The evaluation of our design with respect to its hyperparameters is depicted in Fig. 8. In each of the plots one of the hyperparameters is held constant and the other two are varied across several values. The results imply that the accuracy values decrease when the hypervector density and number of blocks are set to relatively higher or lower values in their respective ranges. Additionally, increasing the dimensionality leads to improved reconstruction results. However, this is not always beneficial as it requires the model to perform more complex calculations when handling larger hypervectors.

5.3. Link prediction task

Similar to GNN, CLOG can also be utilized to represent structural information of knowledge graphs for link prediction. Our general architecture for this task, shown in Fig. 9 and used by Schlichtkrull et al. (2018), applies CLOG as the front-end encoder to acquire graph structure information, and a scoring function (DistMult (Yang et al., 2014)) as the decoder to predict the missing entity. Given HDC’s inherent symbolic property, we only update the entity and relation embedding vectors and maintain the encoding base hypervectors constant during the training process.

As shown in Table 1, we use FB15k-237 (Toutanova & Chen, 2015) and WN18RR (Dettmers et al., 2018) as benchmark datasets to evaluate the link prediction performance. The dimensionality of original embedding vectors and encoded hypervectors is set to 400 bytes and 2000 bytes, respectively. Table 2 demonstrates that CLOG achieves outstanding link prediction accuracy when compared to previous knowledge graph representation models. For FB15K-237, CLOG achieves a Mean Reciprocal Rank (MRR) of 0.356, Hits@10 of 0.526, Hits@3 of 0.393, and Hits@1 of 0.269. On the WN18RR dataset, CLOG records an MRR of 0.465, Hits@10 of 0.521, Hits@3 of 0.487, and Hits@1 of 0.444. More specifically, CLOG demonstrates an average improvement of approximately 26.7% in Mean Reciprocal Rank (MRR) compared to the previous GNN-based model R-GCN (Schlichtkrull et al., 2018). Notably, it achieves competitive results comparable to the advanced GNN-based model CompGCN (Schlichtkrull et al., 2018) and several leading embedding-based models such as InteractE (Vashishth et al.,

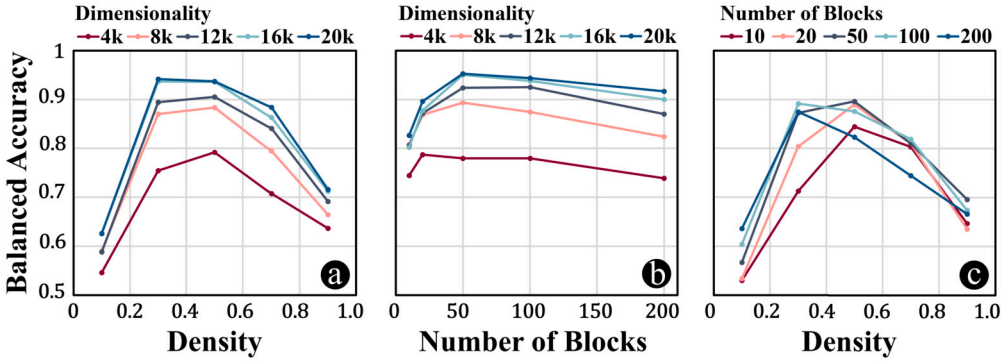

Fig. 8. Reconstruction accuracy plot for CLOG, for different settings of model's three hyperparameters: hypervector dimensionality, density, and number of blocks. Balanced accuracy of the reconstruction is used as the metric for this experiment.

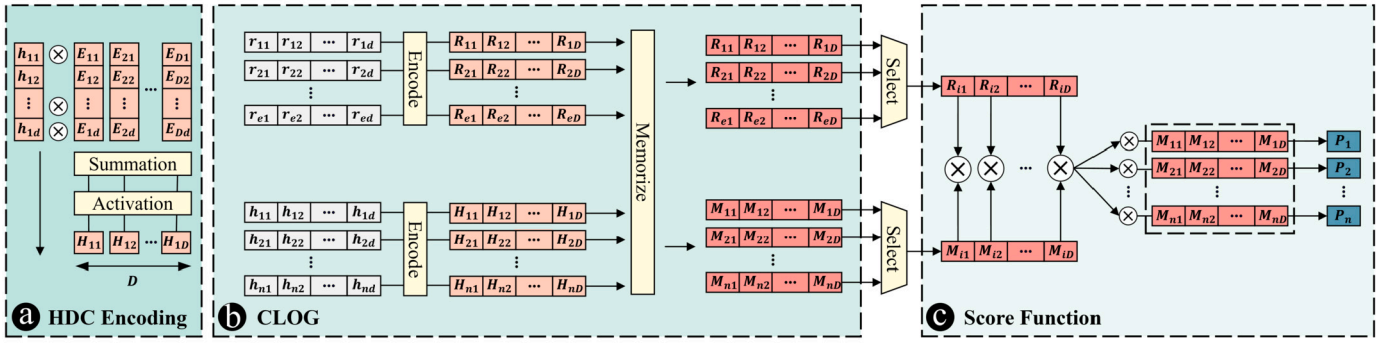

Fig. 9. Overview of the CLOG workflow for knowledge graph tasks. (a) The encoding process for a vector of d dimensions is shown, performed by multiplication with the encoding matrix E, and passing through an activation function, such as *tanh*. (b) The knowledge graph is initiated with low dimensional embeddings of $\boldsymbol{h}$ and $\boldsymbol{r}$ for each entity and relation, respectively. The embeddings are encoded to hypervectors, and passed through CLOG memorization process to obtain the node memory hypervectors $\boldsymbol{M}$. (c) The node memory, along with the relation hypervectors $\boldsymbol{R}$, are passed through a scoring function (DistMult in this instance), which scores them based on the possibility of each triple occurring. The complete model is trained based on the scores, improving the low-dimensional embeddings with each iteration. It is important to note that the HDC encoding matrix is fixed throughout the training process.

Table 1
Link prediction datasets statistics.

Dataset	Entities	Relations	Train	Valid	Test	Avg. degree
WN18RR	40943	11	86835	3034	3134	2.12
FB15K-237	14541	237	272115	17535	20466	18.71

Table 2
Link prediction performance of CLOG and previous models on FB15K-237 and WN18RR datasets. Baseline results are directly taken from the previous works ('-' indicates a missing value). The best values for each metric are highlighted in **bold**.

	FB15K-237				WN18RR			
	MRR	H@10	H@3	H@1	MRR	H@10	H@3	H@1
TransE (Bordes et al., 2013)	0.279	0.441	0.376	0.198	0.243	0.532	0.441	0.043
DistMult (Yang et al., 2014)	0.281	0.446	0.301	0.199	0.444	0.504	0.470	0.412
ComplEx (Trouillon et al., 2016)	0.278	0.450	0.297	0.194	0.449	0.530	0.469	0.409
KBGAN (Cai & Wang, 2017)	0.278	0.458	-	-	0.214	0.472	-	-
ConvKB (Nguyen et al., 2017)	0.243	0.421	-	-	0.249	0.524	-	-
ConvE (Dettmers et al., 2018)	0.312	0.497	0.341	0.225	0.456	0.531	0.470	0.419
R-GCN (Schlichtkrull et al., 2018)	0.164	0.300	0.100	0.181	0.123	0.207	0.137	0.080
RotatE (Sun et al., 2019)	0.338	0.533	0.375	0.241	0.476	**0.571**	0.492	0.428
QuatE (Zhang et al., 2019)	0.331	0.495	0.342	0.221	**0.481**	0.564	**0.500**	0.436
CompGCN (Vashishth et al., 2019)	0.355	**0.535**	0.390	0.264	0.479	0.546	0.494	0.443
InteractE (Vashishth et al., 2020)	0.354	**0.535**	-	0.263	0.463	0.528	-	0.430
CLOG (Proposed Method)	**0.356**	0.526	**0.393**	**0.269**	0.465	0.521	0.487	**0.444**

2020) and RotatE (Sun et al., 2019), surpassing them in multiple metrics. CLOG demonstrates superior performance on FB15k-237 compared to other models, as opposed to WN18RR, for which CLOG performs competitively with other models, outperforming them in Hits@1. This difference can be attributed to dataset properties, where FB15k-237 features over 20 times more relations than WN18RR and approximately 7 times fewer samples per relation. The advancements suggest that CLOG's novel approach to capturing structural relationships within

graphs and its efficient encoding and decoding processes contribute to its enhanced performance in predicting links within knowledge graphs.

6. Conclusion

In this study, we introduced CLOG, an innovative Hyperdimensional Computing (HDC)-based graph representation method. CLOG utilizes HDC's variable binding operator to capture the structural relationships between graph nodes. Our approach, which incorporates block encoding and masking techniques, enables more efficient projection of atomic concepts, such as graph nodes, into hyperspace. This leads to a more relaxed conjunction of neighbors, facilitating improved decoding. The effectiveness of CLOG has been validated through theoretical and empirical experiments, as well as through comparisons with other methods on benchmark link prediction datasets.

Looking ahead, there are several promising directions for further research. Future work could explore the refinement of CLOG's encoding and more specifically decoding processes, the application of the model to more diverse and complex graph types, and the integration of CLOG with other machine learning frameworks to enhance its applicability and performance. The potential for this framework to contribute significantly to the field of graph-based knowledge representation and reasoning is substantial, and continued exploration in this area is expected to yield valuable insights and advancements.

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 National Science Foundation #2127780. Mohsen Imani reports financial support was provided by Semiconductor Research Corp. Mohsen Imani reports financial support was provided by Office of Naval Research #N00014-21-1-2225, #N00014-22-1-2067. Mohsen Imani reports financial support was provided by Air Force Office of Scientific Research #FA9550-22-1-0253.

Data availability

We have utilized public datasets

Acknowledgements

This work was supported in part by National Science Foundation, grant #2127780, Semiconductor Research Corporation (SRC), Office of Naval Research Young Investigator Program Award, 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 Cisco and Xilinx.

Appendix A

Theorem. *Given the graph $G = (V, E)$, with size $|V| = n$ and maximum degree m, the time complexity of decoding the high-dimensional representation of the graph using each of the three models can be quantified as below:*

$$T_1 = O(n^2 D)$$

$$T_2 = O(n^2 D^2(1 + \frac{m}{n} L_{n,m}))$$

$$T_3 = O(n^2 D^2(1 + \frac{1}{n} \sum_{k=0}^{m} \binom{m}{k} (1-\eta)^k \eta^{(m-k)} k L_{n,k}))$$

where $L_{n,k}$ is the number of iterations needed for the resonator network to converge when decomposing k factors with a codebook size of n.

Proof. The decoding process in the previous graph representation method in HDC is discussed in Section 3.2, with Equation (2) reconstructing the node memories, and subsequently measuring their similarity with each node, checking existence of every possible edge. The node reconstruction process involves n operations with D element calculations, leading to a complexity of $O(nD)$. The edge reconstruction includes n^2 similarity checks between hypervectors of dimension D, hence the complexity $O(n^2 D)$. The total decoding complexity as a result is: $T_1 = O(n^2 D)$

The next two methods also include the two steps of node memory and edge reconstruction, the first of which is similar among both methods, with the differences being in the use of blocks and masking. As shown in Equation (11), the node memory reconstruction can be done in $O(nD)$ complexity. For the next step, the intermediate model uses vanilla resonator network to decompose the neighbors from the node memory hypervector, as shown in Equation (5). For each node memory, the algorithm first calculates the product of codebook matrix with its transpose $X_i X_i^T$, which is done in $O(nD^2)$, since X_i has the dimensions $D \times n$. The resulting matrix is then multiplied with the binding of m hypervectors during each iteration, all of which is completed in $O(mD + D^2) = O(D^2)$ time complexity. In total, the algorithm is performed for m factors and runs for $L_{n,m}$ iterations, resulting in $O(nD^2 + mD^2 L_{n,m})$ complexity for each run. The decomposition process should be performed for all n nodes, which leads to the overall time complexity $T_2 = O(nD + n(nD^2 + mD^2 L_{n,m})) = O(n^2 D^2(1 + \frac{m}{n} L_{n,m}))$

CLOG also starts the decoding process by reconstructing the node memory hypervectors in $O(nD)$, similar to previous methods. The next step is executed as shown in Equation (12). The product $H\tilde{H}^T$ is calculated once in $O(nD^2)$, and the process for finding a new guess for each factor is done in $O(mD + D^2) = O(D^2)$. With the use of masking, the algorithm can decompose the blocks with smaller number of factors much faster, as opposed to running for $L_{n,m}$ iterations. The number of factors for each block differs in this method, and can be represented using the binomial distribution; when binding m hypervectors, masked with density of η, the probability of a block having k factors is $q_k = \binom{m}{k}(1-\eta)^k \eta^{(m-k)}$. Hence, the resonator network will decompose k factors, running for $L_{n,k}$ iteration with probability q_k, which leads to a time complexity of $O(nD^2 + D^2 \sum_{k=0}^{m} \binom{m}{k}(1-\eta)^k \eta^{(m-k)} k L_{n,k})$. CLOG performs the factorization for all n nodes, resulting in the final complexity of $T_3 = O(n^2 D^2(1 + \frac{1}{n} \sum_{k=0}^{m} \binom{m}{k}(1-\eta)^k \eta^{(m-k)} k L_{n,k}))$ □

Theorem. *Given the graph $G = (V, E)$, with cardinalities $|V| = n, |E| = e$, and maximum degree m, sensitivity of reconstructed node memory hypervectors can be determined for each of the three methods using the following formulas:*

$$S_1 = \frac{1}{\sqrt{e}} \sqrt{D}$$

$$S_2 = \frac{1}{\sqrt{n}} \sqrt{D}$$

$$S_3 = \frac{1 - \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}{\sqrt{n - \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}} \sqrt{D}$$

$$\text{where} \quad p_{k,m} = \frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}}$$

Proof. Considering Equation (14), we can calculate the sensitivity for detecting existing patterns from non-existing ones in the reconstructed node memory for each method. For the first case, we start the calculation by expanding Equation (2) to its full form:

$$\hat{\boldsymbol{M}}_l = \boldsymbol{G} \odot \boldsymbol{H}_l = 2\boldsymbol{H}_l \odot \sum_{(i,j)\in E} \boldsymbol{H}_i \odot \boldsymbol{H}_j \tag{A.1}$$

$$= \sum_{i \in Nbh(l)} 2\boldsymbol{H}_i + \sum_{i,j \neq l,(i,j) \in E} 2\boldsymbol{H}_l \odot \boldsymbol{H}_i \odot \boldsymbol{H}_j$$

In the expanded form, the initial expression consists of two components. The first one is the unaltered node memory hypervector, which contains the existing patterns in $\hat{\boldsymbol{M}}_l$ in the form of $\boldsymbol{x} \in H$. Meanwhile, the second component comprises the crossterms, which solely contribute to noise.

Given two random hypervectors $\boldsymbol{x}, \boldsymbol{x'} \in H$ such that $\langle \boldsymbol{x}, \boldsymbol{M}_l \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_l \rangle \approx 0$, sensitivity for detecting $\boldsymbol{x}$ from $\boldsymbol{x'}$ in the reconstructed node memory can be derived as below:

$$S_1 = \frac{\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l \rangle] - \mathbf{E}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle]}{\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle]} \tag{A.2}$$

The first expectation in the equation above can be expanded as follows:

$$\begin{aligned} \mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l \rangle] = &\sum_{i \in Nbh(l)} 2\mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{H}_i \rangle] \\ &+ \sum_{i,j \neq l,(i,j) \in E} 2\mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{H}_l \odot \boldsymbol{H}_i \odot \boldsymbol{H}_j \rangle] \end{aligned} \tag{A.3}$$

Since $\boldsymbol{x}$ is selected from H, all the similarities in the second sum are approximately zero, leading to a zero expectation over random selections for $\boldsymbol{x}$. Considering that $\boldsymbol{x}$ has similarity with $\boldsymbol{M}_l$, at least one of the terms in the first sum is non-zero, and $\boldsymbol{x}$ can be similar to only one vector in H, as it is a set of orthogonal hypervectors. Hence, we have:

$$\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l \rangle] = 2 \tag{A.4}$$

We reach a similar form for the second expectation in Equation (A.2) as well, but in this case $\boldsymbol{x'}$ has no similarity with $\boldsymbol{M}_l$, which means that all of the terms in the first sum have approximately zero values. The second sum is also zero, similar to the previous case, since $\boldsymbol{H}_l \odot \boldsymbol{H}_i \odot \boldsymbol{H}_j$ is always orthogonal to any hypervector selected from H. Consequently:

$$\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l \rangle] = 0 \tag{A.5}$$

To calculate the final component in Equation (A.2), we first display the calculation of standard deviation of the similarity between two orthogonal hypervectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, with dimensionality D. Starting from the definition, we have:

$$\begin{aligned} \mathbf{std}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]^2 &= \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle^2] - \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]^2 \\ &= \mathbf{E}[\frac{1}{D^2}(\sum_{i=1}^{D} x_i y_i)^2] - 0 \\ &= \frac{1}{D^2}(\mathbf{E}[\sum_{i=1}^{D} x_i^2 y_i^2] + \mathbf{E}[\sum_{i,j=1,i \neq j}^{D} x_i y_i x_j y_j]) \\ &= \frac{1}{D^2}(D + 0) = \frac{1}{D} \end{aligned} \tag{A.6}$$

leading to the following result: $\mathbf{std}[\langle \boldsymbol{x}, \boldsymbol{y} \rangle] = \frac{1}{\sqrt{D}}$. Returning to the primary issue, the standard deviation can be computed as follows:

$$\begin{aligned} \mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle]^2 = &\sum_{i \in Nbh(H_l)} 4\mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{H}_i \rangle]^2 \\ &+ \sum_{i,j \neq l,(i,j) \in E} 4\mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{H}_l \odot \boldsymbol{H}_i \odot \boldsymbol{H}_j \rangle]^2 \end{aligned} \tag{A.7}$$

As discussed before, all the similarity terms in both summations are between orthogonal hypervectors, and the number of terms are as many as the number of edges e, which leads us to:

$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle] = \sqrt{(\frac{4e}{D})} = 2\sqrt{\frac{e}{D}} \tag{A.8}$$

Combining the calculations, we get the final sensitivity value for the first model: $S_1 = \sqrt{\frac{D}{e}}$

We can follow a similar proof to derive S_2 as well. The node memory hypervector is reconstructed for the second model as below:

$$\begin{aligned} \hat{\boldsymbol{M}}_l &= \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{H}_l) \\ &= \rho^{-1}(\boldsymbol{H}_l \odot \sum_{i=1}^{n} \boldsymbol{H}_i \odot \rho(\prod_{j \in Nbh(i)} \boldsymbol{H}_j)) \\ &= \prod_{i \in Nbh(l)} \boldsymbol{H}_i \\ &+ \sum_{i \neq l}^{n} \rho^{-1}(\boldsymbol{H}_l) \odot \rho^{-1}(\boldsymbol{H}_i) \odot \prod_{j \in Nbh(i)} \boldsymbol{H}_j \end{aligned} \tag{A.9}$$

We need to select the random hypervector for existing and non-existent patterns differently for this model: $\boldsymbol{x}, \boldsymbol{x'} \in \{\prod_{i \in V}^{m_l} \boldsymbol{H}_i\}$ such that $\langle \boldsymbol{x}, \boldsymbol{M}_l \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_l \rangle \approx 0$. The degree of node l is denoted as m_l, which also represents the number of bound hypervectors in M_l. Given the expanded form, we can calculate the needed standard deviation and expectation similar to previous model:

$$\begin{aligned} &\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l \rangle] = 1 \\ &\mathbf{E}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle] = 0 \\ &\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l \rangle] = 2\sqrt{\frac{n}{D}} \end{aligned} \tag{A.10}$$

Note that the number of terms summed together in $\hat{\boldsymbol{M}}_l$ is n in this case, as opposed to e for the first model. As the final result we have $S_2 = \sqrt{\frac{D}{n}}$.

Finally for S_3, which the sensitivity measurement corresponding to CLOG, we have:

$$\begin{aligned} \hat{\boldsymbol{M}}_l &= \rho^{-1}(\boldsymbol{G} \odot \boldsymbol{h}_l) \\ &= \rho^{-1}(\boldsymbol{h}_l \odot \sum_{i=1}^{n} \boldsymbol{h}_i \odot \rho(\boldsymbol{H}_i \odot \prod_{j \in Nbh(i)} \boldsymbol{H}_j)) \\ &= \boldsymbol{H}_l \odot \prod_{i \in Nbh(l)} \boldsymbol{H}_i \\ &+ \sum_{i \neq l}^{n} \rho^{-1}(\boldsymbol{h}_l) \odot \rho^{-1}(\boldsymbol{h}_i) \odot \boldsymbol{H}_i \odot \prod_{j \in Nbh(i)} \boldsymbol{H}_j \end{aligned} \tag{A.11}$$

We select the random hypervector for existing and non-existent patterns as follows: $\boldsymbol{x}, \boldsymbol{x'} \in \{\prod_{i \in V}^{m_l+1} \boldsymbol{H}_i\}$ such that $\langle \boldsymbol{x}, \boldsymbol{M}_l \rangle \gg 0$ and $\langle \boldsymbol{x'}, \boldsymbol{M}_l \rangle \approx 0$. The use of masking introduces a difference in the calculations for this case compared to both previous models, as the masked hypervectors are not necessarily orthogonal.

This can be observed when measuring the similarity between two hypervectors that have been made by binding masked factors. Assume two hypervectors $\boldsymbol{X} = \boldsymbol{X}_1 \odot \cdots \odot \boldsymbol{X}_k$ and $\boldsymbol{Y} = \boldsymbol{Y}_1 \odot \cdots \odot \boldsymbol{Y}_k$, where $\boldsymbol{X}_i$'s and $\boldsymbol{Y}_i$'s are randomly generated masked hypervectors: $\boldsymbol{X}_i = \boldsymbol{x}_i[\boldsymbol{\delta}_{\boldsymbol{x}_i}]$. Denoting $\boldsymbol{X}^{(i)}$ as the ith block of X, the expectation and standard deviation of the similarity between such two hypervectors is derived as follows:

$$\begin{aligned} \mathbf{E}[\langle \boldsymbol{X}, \boldsymbol{Y} \rangle] &= \mathbf{E}[\frac{1}{D}(\sum_{i=1}^{d} \langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)} \rangle)] \\ &= \frac{1}{D}\mathbf{E}[\sum_{i,\forall j : \boldsymbol{\delta}_{\boldsymbol{x}_j}[i]\boldsymbol{\delta}_{\boldsymbol{y}_j}[i]=1}^{d} \langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)} \rangle] \\ &+ \frac{1}{D}\mathbf{E}[\sum_{i,\exists j : \boldsymbol{\delta}_{\boldsymbol{x}_j}[i]\boldsymbol{\delta}_{\boldsymbol{y}_j}[i]=0}^{d} \langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)} \rangle] \end{aligned}$$

$$= \frac{1}{D}(d \times \mathbf{Pr}(\forall j : \boldsymbol{\delta}_{x_j}[i]\boldsymbol{\delta}_{y_j}[i] = 1) \times \frac{D}{d} + 0)$$
$$= (1-\eta)^{2k} \tag{A.12}$$

$$\mathbf{std}[\langle \boldsymbol{X}, \boldsymbol{Y}\rangle]^2 = \mathbf{std}[\frac{1}{D}(\sum_{i=1}^{d}\langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)}\rangle)]^2$$
$$= \frac{1}{D^2}\mathbf{std}[\sum_{i,\forall j:\boldsymbol{\delta}_{x_j}[i]\boldsymbol{\delta}_{y_j}[i]=1}^{d}\langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)}\rangle]^2$$
$$+ \frac{1}{D^2}\mathbf{std}[\sum_{i,\exists j:\boldsymbol{\delta}_{x_j}[i]\boldsymbol{\delta}_{y_j}[i]=0}^{d}\langle \boldsymbol{X}^{(i)}, \boldsymbol{Y}^{(i)}\rangle]^2$$
$$= \frac{1}{D^2}(0 + d \times \mathbf{Pr}(\exists j : \boldsymbol{\delta}_{x_j}[i]\boldsymbol{\delta}_{y_j}[i] = 0) \times \frac{D}{d})$$
$$= \frac{1}{D}(1-(1-\eta)^{2k}) \tag{A.13}$$

Having the mentioned measurements in mind, we start deriving the sensitivity in the case of CLOG:

$$\mathbf{E}[\langle \boldsymbol{x}, \hat{\boldsymbol{M}}_l\rangle] = \mathbf{E}[\langle \boldsymbol{x}, \boldsymbol{H}_l \odot \prod_{i\in N bh(l)} \boldsymbol{H}_i\rangle]$$
$$+ \sum_{i\neq l}^{n}\mathbf{E}[\langle \boldsymbol{x}, \rho^{-1}(\boldsymbol{h}_l \odot \boldsymbol{h}_i) \odot \boldsymbol{H}_i \odot \prod_{j\in N bh(i)} \boldsymbol{H}_j\rangle]$$
$$= 1 + 0 = 1 \tag{A.14}$$

Note that $\boldsymbol{x}$ is similar to $\boldsymbol{H}_l \odot \prod_{i\in N bh(l)} \boldsymbol{H}_i$ since it is selected as an existing pattern in the node memory hypervector. It is also not possible for any of the terms in the following summation to have non-zero values, as binding of the hypervector $\rho^{-1}(\boldsymbol{h}_l \odot \boldsymbol{h}_i)$ prevents it.

As for $\boldsymbol{x'}$, it can contain factors that are also a part of the term $\boldsymbol{H}_l \odot \prod_{i\in N bh(l)} \boldsymbol{H}_i$, depending on how it is selected. The probability of these two hypervectors having k non-similar factors (out of $m+1$ total factors), denoted as $p_{k,m}$, is shown below:

$$p_{k,m} = \frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}} \tag{A.15}$$

which is derived by selecting the k non-similar factors out of $m+1$, and the remaining factors from all the other $n-(m+1)$ factors that are not in $\boldsymbol{H}_l \prod_{i\in N bh(l)} \boldsymbol{H}_i$, and then normalizing with the total number of selections for $m+1$ factors out of all the possible n factors.

Consequently, the similarity between $\boldsymbol{x'}$ and $\hat{\boldsymbol{M}}_l$ can be measured by getting an expectation over all the possible values of k as below:

$$\mathbf{E}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l\rangle] = \mathbf{E}[\langle \boldsymbol{x'}, \boldsymbol{H}_l \odot \prod_{i\in N bh(l)} \boldsymbol{H}_i\rangle]$$
$$+ \sum_{i\neq l}^{n}\mathbf{E}[\langle \boldsymbol{x'}, \rho^{-1}(\boldsymbol{h}_l \odot \boldsymbol{h}_i) \odot \boldsymbol{H}_i \odot \prod_{j\in N bh(i)} \boldsymbol{H}_j\rangle]$$
$$= \sum_{k=0}^{m+1}\frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}}(1-\eta)^{2k} + 0$$
$$= \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k} \tag{A.16}$$

To calculate the standard deviation of the similarity we also have:

$$\mathbf{std}[\langle \boldsymbol{x'}, \hat{\boldsymbol{M}}_l\rangle]^2 = \mathbf{std}[\langle \boldsymbol{x'}, \boldsymbol{H}_l \odot \prod_{i\in N bh(l)} \boldsymbol{H}_i\rangle]^2$$
$$+ \sum_{i\neq l}^{n}\mathbf{std}[\langle \boldsymbol{x'}, \rho^{-1}(\boldsymbol{h}_l \odot \boldsymbol{h}_i) \odot \boldsymbol{H}_i \odot \prod_{j\in N bh(i)} \boldsymbol{H}_j\rangle]^2$$
$$= \frac{1}{D}\sum_{k=0}^{m+1}\frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}}(1-(1-\eta)^{2k})$$
$$+ (n-1)\times\frac{1}{D}$$
$$= \frac{1}{D}\sum_{k=0}^{m+1}\frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}}$$
$$- \frac{1}{D}\sum_{k=0}^{m+1}\frac{\binom{m+1}{k}\binom{n-(m+1)}{k}}{\binom{n}{m+1}}(1-\eta)^{2k}$$
$$+ \frac{1}{D}(n-1)$$
$$= \frac{1}{D}(n - \sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}) \tag{A.17}$$

Combining all the computations, we arrive at the outcome presented in Equation (15): $S_3 = \frac{1-\sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}{\sqrt{n-\sum_{k=0}^{m+1} p_{k,m}(1-\eta)^{2k}}}\sqrt{D}$. □

References

Arora, S. (2020). A survey on graph neural networks for knowledge graph completion. arXiv preprint, arXiv:2007.12374.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., et al. (2000). Gene ontology: Tool for the unification of biology. *Nature Genetics*, *25*, 25–29.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on management of data* (pp. 1247–1250).

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems*, *26*.

Cai, L., & Wang, W. Y. (2017). Kbgan: Adversarial learning for knowledge graph embeddings. arXiv preprint, arXiv:1711.04071.

Chen, H., Issa, M., Ni, Y., & Imani, M. (2022). DARL: Distributed Reconfigurable Accelerator for Hyperdimensional Reinforcement Learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (pp. 1–9). IEEE.

Chen, H., Zakeri, A., Wen, F., Barkam, H. E., & Imani, M. (2023). Hypergraf: Hyperdimensional graph-based reasoning acceleration on fpga. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)* (pp. 34–41). IEEE.

Chen, H., Ni, Y., Zakeri, A., Zhuowen, Z., Yun, S., Wen, F., Khaleghi, B., Srinivasa, N., Latapie, H., & Imani, M. (2024). HDReason: Algorithm-Hardware Codesign for Hyperdimensional Knowledge Graph Reasoning. arXiv preprint, arXiv:2403.05763.

Chen, P., Jiao, R., Liu, J., Liu, Y., & Lu, Y. (2022). Interpretable graph transformer network for predicting adsorption isotherms of metal–organic frameworks. *Journal of Chemical Information and Modeling*, *62*, 5446–5456.

Chen, X., Jia, S., & Xiang, Y. (2020). A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, *141*, Article 112948.

Cheng, D., Yang, F., Xiang, S., & Liu, J. (2022). Financial time series forecasting with multi-modality graph neural network. *Pattern Recognition*, *121*, Article 108218.

Clarkson, K. L., Ubaru, S., & Yang, E. (2023). Capacity analysis of vector symbolic architectures. arXiv preprint, arXiv:2301.10352.

Dettmers, T., Minervini, P., Stenetorp, P., & Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence: Vol. 32*.

Dwivedi, V. P., & Bresson, X. (2020). A generalization of transformer networks to graphs. arXiv preprint, arXiv:2012.09699.

Fellbaum, C. (2010). *Wordnet, in: Theory and applications of ontology: Computer applications*. Springer (pp. 231–243).

Feng, A., You, C., Wang, S., & Tassiulas, L. (2022). Kergnns: Interpretable graph neural networks with graph kernels. In *Proceedings of the AAAI Conference on Artificial Intelligence: Vol. 36* (pp. 6614–6622).

Frady, E. P., Kleyko, D., & Sommer, F. T. (2018). A theory of sequence indexing and working memory in recurrent neural networks. *Neural Computation*, *30*, 1449–1513.

Frady, E. P., Kent, S. J., Olshausen, B. A., & Sommer, F. T. (2020). Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures. *Neural Computation*, *32*, 2311–2331.

Frady, E. P., Sanborn, S., Shrestha, S. B., Rubin, D. B. D., Orchard, G., Sommer, F. T., & Davies, M. (2022). Efficient neuromorphic signal processing with resonator neurons. *Journal of Signal Processing Systems*, *94*, 917–927.

Gao, Z., Jiang, C., Zhang, J., Jiang, X., Li, L., Zhao, P., Yang, H., Huang, Y., & Li, J. (2023). Hierarchical graph learning for protein–protein interaction. *Nature Communications*, *14*, 1093.

Gayler, R. (1998). *Multiplicative binding, representation operators and analogy*.

Gayler, R. W., & Levy, S. D. (2009). A distributed basis for analogical mapping. In *New Frontiers in Analogy Research; Proc. of 2nd Intern. Analogy Conf.: Vol. 9*.

Ge, L., & Parhi, K. K. (2022). Applicability of hyperdimensional computing to seizure detection. *IEEE Open Journal of Circuits and Systems*, *3*, 59–71.

Gentner, D., & Smith, L. A. (2013). Analogical learning and reasoning. In *The Oxford handbook of cognitive psychology* (pp. 668–681).

Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 855–864).

Hersche, M., Karunaratne, G., Cherubini, G., Benini, L., Sebastian, A., & Rahimi, A. (2022). Constrained few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9057–9067).

Hersche, M., Zeqiri, M., Benini, L., Sebastian, A., & Rahimi, A. (2023). A neuro-vector-symbolic architecture for solving raven's progressive matrices. *Nature Machine Intelligence*, 1–13.

Huang, C., Li, M., Cao, F., Fujita, H., Li, Z., & Wu, X. (2022). Are graph convolutional networks with random weights feasible? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*, 2751–2768.

Huang, Q., Yamada, M., Tian, Y., Singh, D., & Chang, Y. (2022). Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *35*, 6968–6972.

Imani, M., Zakeri, A., Chen, H., Kim, T., Poduval, P., Lee, H., Kim, Y., Sadredini, E., & Imani, F. (2022). Neural computation for robust and holographic face detection. In *2022 59th ACM/IEEE Design Automation Conference* (pp. 31–36).

Kanerva, P. (1996). Binary spatter-coding of ordered k-tuples. In *International Conference on Artificial Neural Networks* (pp. 869–873). Springer.

Kang, J., Zhou, M., Bhansali, A., Xu, W., Thomas, A., & Rosing, T. (2022). Relhd: A graph-based learning on fefet with hyperdimensional computing. In *2022 IEEE 40th International Conference on Computer Design (ICCD)* (pp. 553–560). IEEE.

Kent, S. J., Frady, E. P., Sommer, F. T., & Olshausen, B. A. (2020). Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods. *Neural Computation*, *32*, 2332–2388.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint, arXiv:1609.02907.

Lee, H., Kim, J., Chen, H., Zeira, A., Srinivasa, N., Imani, M., & Kim, Y. (2023). Comprehensive Integration of Hyperdimensional Computing with Deep Learning towards Neuro-Symbolic AI. In *2023 60th ACM/IEEE Design Automation Conference (DAC)* (pp. 1–6).

Li, J., Yu, Z., Zhu, Z., Chen, L., Yu, Q., Zheng, Z., Tian, S., Wu, R., & Meng, C. (2023). Scaling up dynamic graph representation learning via spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence: Vol. 37* (pp. 8588–8596).

Li, J., Zheng, R., Feng, H., & Zhuang, X. (2023). Permutaion equivariant graph framelets for heterophilous semi-supervised learning. arXiv preprint, arXiv:2306.04265.

Li, M., Zhang, L., Cui, L., Bai, L., Li, Z., & Wu, X. (2023). Blog: Bootstrapped graph representation learning with local and global regularization for recommendation. *Pattern Recognition*, *144*, Article 109874.

Liu, L., Wen, G., Cao, P., Hong, T., Yang, J., Zhang, X., & Zaiane, O. R. (2023). Braintgl: A dynamic graph representation learning model for brain network analysis. *Computers in Biology and Medicine*, *153*, Article 106521.

Moin, A., Zhou, A., Rahimi, A., Menon, A., Benatti, S., Alexandrov, G., Tamakloe, S., Ting, J., Yamamoto, N., Khan, Y., et al. (2021). A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics*, *4*, 54–63.

Nguyen, D. Q., Nguyen, T. D., Nguyen, D. Q., & Phung, D. (2017). A novel embedding model for knowledge base completion based on convolutional neural network. arXiv preprint, arXiv:1712.02121.

Ni, Y., Issa, M., Abraham, D., Imani, M., Yin, X., & Imani, M. (2022). Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (pp. 1141–1146).

Ni, Y., Lesica, N., Zeng, F., & Imani, M. (2022). Neurally-Inspired Hyperdimensional Classification for Efficient and Robust Biosignal Processing. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (pp. 1–9).

Ni, Y., Zou, Z., Huang, W., Chen, H., Chung, W. Y., Cho, S., Krishnan, R., & Imani, M. (2024). HEAL: Brain-inspired Hyperdimensional Efficient Active Learning. arXiv preprint, arXiv:2402.11223.

Nunes, I., Heddes, M., Givargis, T., Nicolau, A., & Veidenbaum, A. (2022). Graphhd: Efficient graph classification using hyperdimensional computing. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1485–1490). IEEE.

Park, N., Kan, A., Dong, X. L., Zhao, T., & Faloutsos, C. (2019). Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 596–606).

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 701–710).

Plate, T., et al. (1991). Holographic reduced representations: Convolution algebra for compositional distributed representations. In *IJCAI* (pp. 30–35). Citeseer.

Poduval, P., Alimohamadi, H., Zakeri, A., Imani, F., Najafi, M. H., Givargis, T., & Imani, M. (2022). Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning. *Frontiers in Neuroscience*, *16*, Article 757125.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., & Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, *35*, 14501–14515.

Renner, A., Supic, L., Danielescu, A., Indiveri, G., Olshausen, B. A., Sandamirskaya, Y., Sommer, F. T., & Frady, E. P. (2022). Neuromorphic visual scene understanding with resonator networks. arXiv preprint, arXiv:2208.12880.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference* (pp. 593–607). Springer.

Sun, Z., Deng, Z.-H., Nie, J.-Y., & Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. arXiv preprint, arXiv:1902.10197.

Thakoor, S., Tallec, C., Azar, M. G., Azabou, M., Dyer, E. L., Munos, R., Veličković, P., & Valko, M. (2021). Large-scale representation learning on graphs via bootstrapping. arXiv preprint, arXiv:2102.06514.

Toutanova, K., & Chen, D. (2015). Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality* (pp. 57–66).

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., & Bouchard, G. (2016). Complex embeddings for simple link prediction. In *International Conference on Machine Learning* (pp. 2071–2080). PMLR.

Vashishth, S., Sanyal, S., Nitin, V., & Talukdar, P. (2019). Composition-based multi-relational graph convolutional networks. arXiv preprint, arXiv:1911.03082.

Vashishth, S., Sanyal, S., Nitin, V., Agrawal, N., & Talukdar, P. (2020). Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence: Vol. 34* (pp. 3009–3016).

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., et al. (2017). Graph attention networks. *Stat*, *1050*, Article 10-48550.

Wang, Z., Li, Z., Leng, J., Li, M., & Bai, L. (2022). Multiple pedestrian tracking with graph attention map on urban road scene. *IEEE Transactions on Intelligent Transportation Systems*, *24*(8), 8567–8579.

Wu, Z., Lin, X., Lin, Z., Chen, Z., Bai, Y., & Wang, S. (2023). Interpretable graph convolutional network for multi-view semi-supervised learning. *IEEE Transactions on Multimedia*, *25*, 8593–8606.

Yang, B., Yih, W-t., He, X., Gao, J., & Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint, arXiv:1412.6575.

Zamini, M., Reza, H., & Rabiei, M. (2022). A review of knowledge graph completion. *Information*, *13*, 396.

Zhang, D., Yuan, Z., Liu, H., Xiong, H., et al. (2022). Learning to walk with dual agents for knowledge graph reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence: Vol. 36* (pp. 5932–5941).

Zhang, S., Tay, Y., Yao, L., & Liu, Q. (2019). Quaternion knowledge graph embeddings. *Advances in Neural Information Processing Systems*, *32*.

Zou, Z., Alimohamadi, H., Zakeri, A., Imani, F., Kim, Y., Najafi, M. H., & Imani, M. (2022). Memory-inspired spiking hyperdimensional network for robust online learning. *Scientific Reports*, *12*, 7641.