VilLain: Self-Supervised Learning on Homogeneous Hypergraphs without Features via Virtual Label Propagation

Geon Lee
Korea Advanced Institute of Science and Technology
Seoul, Republic of Korea
geonlee0325@kaist.ac.kr

Soo Yong Lee
Korea Advanced Institute of Science and Technology
Seoul, Republic of Korea
syleetolow@kaist.ac.kr

Kijung Shin
Korea Advanced Institute of Science and Technology
Seoul, Republic of Korea
kijungs@kaist.ac.kr

ABSTRACT

Group interactions arise in various scenarios in real-world systems: collaborations of researchers, co-purchases of products, and discussions in online Q&A sites, to name a few. Such higher-order relations are naturally modeled as hypergraphs, which consist of hyperedges (i.e., any-sized subsets of nodes). For hypergraphs, the challenge to learn node representation when features or labels are not available is imminent, given that (a) most real-world hypergraphs are not equipped with external features while (b) most existing approaches for hypergraph learning resort to additional information. Thus, in this work, we propose VilLain, a novel self-supervised hypergraph representation learning method based on the propagation of virtual labels (v-labels). Specifically, we learn for each node a sparse probability distribution over v-labels as its feature vector, and we propagate the vectors to construct the final node embeddings. Inspired by higher-order label homogeneity, which we discover in real-world hypergraphs, we design novel self-supervised loss functions for the v-labels to reproduce the higher-order structure-label pattern. We demonstrate that VilLain is: **(a) Requirement-free**: learning node embeddings without relying on node labels and features, **(b) Versatile**: giving embeddings that are not specialized to specific tasks but generalizable to diverse downstream tasks, and **(c) Accurate**: more accurate than its competitors for node classification, hyperedge prediction, node clustering, and node retrieval tasks. Our code and dataset are available at https://github.com/geon0325/VilLain.

CCS CONCEPTS

• **Computing methodologies** → **Unsupervised learning**; • **Information systems** → **Data mining**; **Social networks**.

KEYWORDS

Hypergraph, Representation Learning, Self-Supervised Learning

ACM Reference Format:
Geon Lee, Soo Yong Lee, and Kijung Shin. 2024. VilLain: Self-Supervised Learning on Homogeneous Hypergraphs without Features via Virtual Label Propagation. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645454

WWW '24, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0171-9/24/05.
https://doi.org/10.1145/3589334.3645454

1 INTRODUCTION

In many real-world complex systems, interactions often occur in groups: research collaborations, email communications, group discussions, and protein interactions, to name a few. Representing such group interactions (i.e., higher-order relationships) as edges in an ordinary pairwise graph impairs the semantics of the interactions, often leading to considerable information loss [12, 38, 78].

Hypergraphs address the limitations of ordinary graphs by modeling group interactions as hyperedges, the non-empty subsets of nodes. Specifically, the flexibility in hyperedge sizes enables each hyperedge to naturally represent an interaction among any number of nodes. Hypergraphs are used to model data from various fields (refer to a survey [36]), including bioinformatics [31], social network analysis [75], circuit design [32], and computer vision [29, 33, 68]. Notably, hypergraph modeling has demonstrated its effectiveness over ordinary graphs in diverse applications, such as recommendation [69, 70], medical prediction [5], and crime prediction [43].

A popular approach for analyzing such complex relations is to learn node embeddings (i.e., vector representations of nodes) through *self-supervision.* In hypergraphs, self-supervised learning has been applied for node classification [27, 35, 66], hyperedge prediction [64, 84], recommendation [71, 81], and user location prediction in social media [75]. Self-supervised learning enjoys several key advantages. It does not require external node labels, which are scarce in many real-world scenarios due to substantial costs in their acquisition [26]. Moreover, the learned embeddings often demonstrate considerable *versatility*, maintaining their utility across a broad range of tasks.

Many self-supervised node embedding methods require external features. Hypergraph Neural Networks (HNNs) [10, 16, 20, 28, 35, 66] and Graph Neural Networks (GNNs) [24, 34, 52, 62, 63, 72, 85], for instance, heavily rely on the external node features. As such, most of them are only tested on attributed benchmark datasets [17, 25, 56, 58, 76], and their performances strongly depend on the feature quality [14, 18, 44, 49].

Despite their usefulness, external features are often entirely or partially missing in real-world hypergraphs [9, 14, 18, 55, 77, 84]. In fact, only 3.03% of the graphs at a popular graph database are given with node features [56], [1] and none of the hypergraphs at the largest hypergraph database is attributed. [2] Such a problem, in combination with the issue of label scarcity, poses an imminent challenge for hypergraph representation learning.

While some self-supervised approaches do not require external features, their embeddings are hardly versatile. Some link prediction

[1]Out of 6,659 graph datasets, 202 are given with node attributes.
[2]https://www.cs.cornell.edu/~arb/data/

HNNs and GNNs leverage the structural or identity features [6, 64, 80, 82, 86] without the external ones, and random walk (RW) [23, 27, 53]- or matrix factorization (MF) [50, 54, 60]-based methods (i.e. Hyper2Vec) only need graph structure for their node embeddings. However, they arguably only preserve structural properties, since their input and objective functions are *solely structural.* Such models are, thus, less applicable to tasks where the importance of structural property is less prominent, such as node classification.

In this paper, we aim to learn versatile node embeddings for hypergraphs without relying on external labels or features. To this end, we propose VilLain (**Vi**rtua**l** **La**bel Propagat**i**o**n**). VilLain constructs for each node a sparse probability distribution over virtual labels (v-labels) as its feature. The probabilistic v-label assignment vectors are propagated to construct the final node embeddings. At each propagation step, the v-labels are optimized with novel self-supervised loss functions, inspired by higher-order label homogeneity in real-world hypergraphs. Thus, VilLain learns potential (higher-order) structure-label relationships, beyond purely structural properties.

Through extensive experiments using eight real-world hypergraphs and three downstream tasks (specifically, node classification, node retrieval, node clustering, and hyperedge prediction), we demonstrate the superiority of VilLain over 15 baseline approaches. We summarize its strengths as follows:

- **Minimum Requirements:** VilLain learns node embeddings without any supervision (e.g., node labels) or extra information (e.g., node features and the number of labels).
- **Versatile Embedding:** VilLain learns general-purpose node embeddings that are not specialized to specific tasks but generalized to diverse downstream tasks.
- **Accurate Embedding:** VilLain achieves up to 71.6%, 72.3%, and 6.7% better accuracy than unsupervised and (semi-)supervised baseline approaches for node classification, node retrieval, and hyperedge prediction tasks, respectively.

2 RELATED WORK

In this section, we briefly review related works on node representation learning, focusing on learning without labels or features.

Node embedding with propagation. Propagation has been widely applied and shown effective for both hypergraph and graph representation learning. GNNs typically have each node propagate its features to the direct neighbors [8, 22, 34], whereas for HNNs, the propagation is conducted on hypergraph structure. Specifically, HGNN [20] has each node propagate to its hyperedges, where the node features are aggregated and propagated back to the nodes that belong to the hyperedges. HNHN [16] uses non-linear aggregation functions to update both node and hyperedge embeddings, alternatingly. AllSet [10] uses permutation-invariant functions to propagate on hyperedges. Other simplified GNNs [11, 15, 21, 67] first learn soft label vectors from feature vectors, which are propagated to learn the final node embeddings. Note that all the described methods require external labels or features.

Node embedding without external labels. Self-supervision has been widely adopted for representation learning without external labels. Self-supervised HNNs and GNNs often utilize contrastive losses. Given both original and perturbed features or structures, the models maximize the mutual information between them [35, 63, 85]. For hypergraphs, HyperGCL [66] uses node- and hyperedge-level perturbation, and TriCL [35] conducts tri-directional contrasts that maximize the agreement between two augmented views of nodes, groups, and memberships. Intuitively, such self-supervised loss functions are designed to learn node embeddings that denoise the input features and structure. It, then, implies that these self-supervised models can only learn structural properties if their input node features are random or structural.

Given random walk sequences, RW-based embedding methods [23, 27, 53] typically use Skip-Gram [47] to optimize the embeddings to maximize the likelihood of the visited nodes. MF-based approaches [50, 54, 60], on the other hand, factorize proximity matrices into low-rank matrices. As such, most RW- and MF-based embedding methods specifically preserve structural proximity.

Node embedding without external features. If external features are not available, HNNs and GNNs require derived features for their prediction. For structural prediction, some models have leveraged only structural information as the input features [6, 14, 64, 80, 82, 86]. Specifically, structural [4, 6, 14, 80], positional [14, 41, 65], and identity [1, 57, 64, 79, 82, 83, 86] encoding methods have been developed. Such encoding methods generally aim to enhance model expressivity beyond 1-WL test [72]. On the other hand, the majority of RW- and MF-based approaches do not require any features or labels [23, 27, 50, 53, 54, 60]. It is, however, worth noting that all the described methods *over-emphasize structural properties*, since their features and objective loss functions are solely structural. Thus, predictions from their embeddings hardly generalize to less structure-dependent tasks, such as node classification.[3]

Relating VilLain to the prior works. In comparison to (hyper) graph learning models without external features or labels, we present the novelty of VilLain in the subsequent sections as follows:

- **Novel Self-Supervised Loss:** Only VilLain has loss function that learns *beyond structural information* for embedding versatility.
- **Novel Input Feature Learning:** VilLain's motivation and mechanism of feature learning are distinct from the prior methods.

3 PROBLEM STATEMENT

In this section, we formulate hypergraph representation learning without features or labels. A hypergraph $G = (V, E)$ consists of a set of nodes $V = \{v_1, \cdots, v_{|V|}\}$ and a set of hyperedges $E = \{e_1, \cdots, e_{|E|}\}$. Each hyperedge $e_j \in E$ is a non-empty subset of nodes, i.e., $\emptyset \subsetneq e_j \subseteq V$. In the incidence matrix $\mathbf{H} \in \{0,1\}^{|V|\times|E|}$ of G, $\mathbf{H}_{ij} = 1$, if $v_i \in e_j$, and $\mathbf{H}_{ij} = 0$ otherwise.

Given a hypergraph $G = (V, E)$, the objective of self-supervised hypergraph representation learning is to learn a node embedding $\mathbf{Z}_i \in \mathbb{R}^d$ of each node $v_i \in V$, or equivalently, a node embedding matrix $\mathbf{Z} \in \mathbb{R}^{|V|\times d}$ that captures meaningful proximity between nodes in G. Specifically, we aim to learn node embeddings that are generally useful for various tasks (e.g., node classification and hyperedge prediction), without relying on any kind of supervision (e.g., ground-truth semantic labels or even the number of unique labels) or external information (e.g., node attributes).

[3]See the low performances of such methods (e.g. Hyper2Vec, HyperGCL) in Table 2.

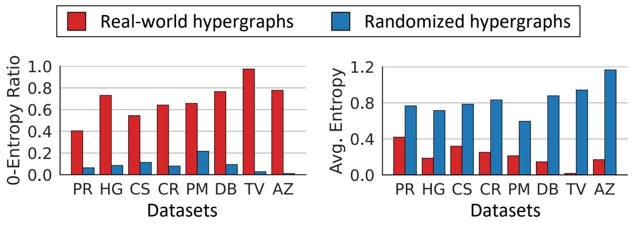

Figure 1: Hyperedges in real-world hypergraphs (statistics in Table 1) exhibit label homogeneity (Obs. 1).

4 MOTIVATING OBSERVATIONS

In this section, we present our observation in real-world hypergraphs, which motivate the design of VilLain in Section 5. Inspired by pervasive *homophily* [2, 46] in real-world graphs, we postulate that hypergraphs also exhibit a similar tendency. For example, researchers from the same area tend to co-author a paper, and e-mails are likely to be exchanged within the same department. To substantiate this hypothesis, we examine label homogeneity in eight different real-world hypergraphs.

Using the ground-truth node labels, for each hyperedge, we measure the entropy of its soft label assignment vector, which is obtained by averaging the label assignment one-hot vectors of the nodes in the hyperedge. If the entropy is 0, all nodes in the hyperedge are labeled identically (high homogeneity). The higher the entropy is, the more diverse labels the nodes in the hyperedge have (low homogeneity). As shown in Figure 1, the entropy in real-world hypergraphs tends to be lower than that in hypergraphs that are randomized by a random hypergraph generator, HyperCL [37]. Specifically, the ratio of the hyperedges with entropy 0 is much higher in real-world hypergraphs than in the randomized hypergraphs, and the average entropy is lower in real-world hypergraphs than in the randomized hypergraphs.

Observation 1. *Hyperedges in real-world hypergraphs exhibit label homogeneity, i.e., they tend to contain the same labeled nodes.*

In addition, we examine higher-order homogeneity in real-world hypergraphs. To this end, we measure the entropy of the higher-order label assignment vectors (or ℓ-step labels in short) of hyperedges. For each $\ell \geq 0$, the ℓ-step label of a hyperedge is obtained by averaging the ℓ-step labels of the nodes in it. The ℓ-step label of each node is given if $\ell = 0$, or obtained by averaging $(\ell - 1)$-step labels of the incident hyperedges (the detailed procedure can be found in Section 5.1). Figure 2 demonstrates that (a) the entropy of 50-step labels of hyperedges in a real-world hypergraph (spec., Trivago) is lower than those in the randomized counterpart, and (b) regardless of the step count ℓ, hyperedges in the real-world hypergraph exhibit higher homogeneity than those in the randomized hypergraph. These findings provide concrete evidence supporting the presence of higher-order homogeneity in real-world hypergraphs.

Observation 2. *Real-world hypergraphs exhibit higher-order label homogeneity, i.e., the node labels in each hyperedge tend to be homogeneous even after multiple steps of propagation.*

5 PROPOSED METHOD

In this section, we propose VilLain (Figure 3), a self-supervised node representation learning method for hypergraphs. Notably, VilLain does not require external labels or features.

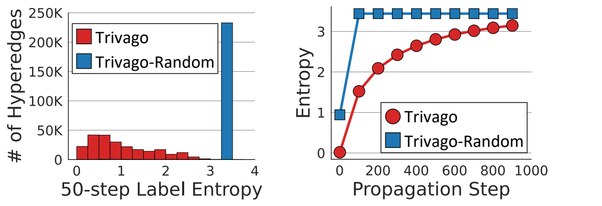

Figure 2: Real-world hypergraphs exhibit higher-order label homogeneity (Obs. 2).

5.1 VilLain: Virtual Label Propagation

We first present how VilLain obtains node embeddings through virtual label (v-labels) propagation, *without external features.*

Virtual Labels. Since node labels or features are not given, VilLain assumes the presence of d v-labels and leverages the soft v-label assignment vector of each node as its learnable feature. Specifically, VilLain employs a learnable matrix $\widetilde{\mathbf{X}} \in \mathbb{R}^{|V| \times d}$ where each i^{th} row $\widetilde{\mathbf{X}}_i$ is used to obtain the soft assignment vector $\mathbf{X}_i^{(0)} \in [0,1]^d$ of the node v_i to d v-labels as follows:

$$\mathbf{X}_{ij}^{(0)} = \frac{e^{(\widetilde{\mathbf{X}}_{ij} + g_j)}}{\sum_{j'=1}^{d} e^{(\widetilde{\mathbf{X}}_{ij'} + g_{j'})}}, \quad \text{for } j = 1, \cdots, d, \tag{1}$$

where $g_j = -\log(\log(\frac{1}{u_i}))$ is random noise and $u_i \sim \text{Uniform}(0,1)$. The above equation transforms the vector into a probability vector and encourages it to be biased towards a single v-label. As described later, the v-label assignment vectors are optimized to reproduce higher-order label homogeneity (Observations 1 and 2).

Hypergraph V-label Propagation. After obtaining the v-label matrix $\mathbf{X}^{(0)}$, VilLain conducts v-label propagation on the input hypergraph to obtain $\mathbf{X}^{(\ell)}$. At each step, v-labels are propagated alternatingly between nodes and hyperedges. Specifically, the v-label assignment matrices of hyperedges and nodes at step ℓ are:

$$\mathbf{Y}^{(\ell)} = \mathbf{D}_E^{-1}\mathbf{H}^T\mathbf{X}^{(\ell-1)} \quad \text{and} \quad \mathbf{X}^{(\ell)} = \mathbf{D}_V^{-1}\mathbf{H}\mathbf{Y}^{(\ell)}, \tag{2}$$

where $\mathbf{D}_V$ and $\mathbf{D}_E$ are the diagonal matrices with node degrees and hyperedge sizes, respectively. To capture higher-order dependencies among nodes, VilLain computes node embeddings $\mathbf{Z} \in [0,1]^{|V| \times d}$ by averaging the v-label assignment vectors obtained at propagation steps $1, \cdots, k'$:

$$\mathbf{Z} = \frac{1}{k'} \sum_{\ell=1}^{k'} \mathbf{X}^{(\ell)}. \tag{3}$$

Namely, the embedding $\mathbf{Z}_i$ of node v_i is a probability vector averaging its v-label assignment vector at each step.

Multi-V-label Propagation. In real-world hypergraphs, nodes may have multiple labels, each representing different aspects. For instance, in a social network, socioeconomic status and political inclination can both serve as labels, albeit their independent homogeneity w.r.t. hypergraph topology. The same goes for the number of labels. Learning a single set of v-labels, then, can be *insufficient* to capture their complex structure-label patterns.

Thus, VilLain learns multi-v-labels for the final node embedding $\mathbf{Z}^*$. Specifically, we partition the d-dimensional embedding space into D subspaces of potentially different dimensions, allowing for independent v-label propagation within each subspace. Then,

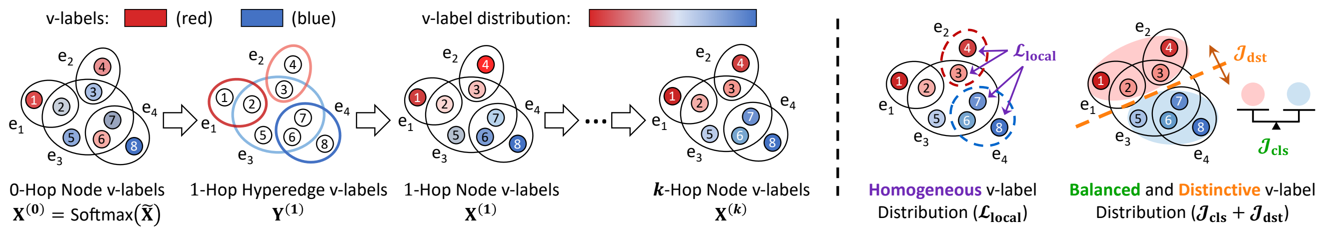
v-labels: (red) (blue)
v-label distribution:
0-Hop Node v-labels
$X^{(0)} = \text{Softmax}(\tilde{X})$
1-Hop Hyperedge v-labels
$Y^{(1)}$
1-Hop Node v-labels
$X^{(1)}$
k-Hop Node v-labels
$X^{(k)}$
Homogeneous v-label Distribution ($\mathcal{L}_{\text{local}}$)
Balanced and Distinctive v-label Distribution ($\mathcal{J}_{\text{cls}} + \mathcal{J}_{\text{dst}}$)

Figure 3: (Left) Two v-labels (red and blue) are propagated between nodes and hyperedges on a hypergraph (Sec. 5.1). Note that hyperedges are colored to indicate $Y^{(1)}$. (Right) By minimizing the proposed local and global losses, the v-label distributions are learned to exhibit higher-order label homogeneity while being balanced and distinctive at each propagation step (Sec. 5.2).

VilLain concatenates the outputs from each subspace as follows:

$$Z_i^* = \left[Z_i^{\langle 1 \rangle} \parallel Z_i^{\langle 2 \rangle} \parallel \cdots \parallel Z_i^{\langle D \rangle} \right],$$

where $\parallel$ is the concatenation operation, and $Z_i^{\langle \cdot \rangle}$ is the embedding obtained from each subspace.

5.2 Self-Supervision Objectives

The learning objectives of VilLain are designed to reproduce higher-order label homogeneity by effectively capturing structural properties and also potential higher-order structure-label relationships. Recall that the entries of the matrix $\tilde{X}$ are the only learnable parameters in VilLain that the objective function updates.

Capturing Local Information. Motivated by Observations 1 and 2 in Section 4, we design an objective to capture the higher-order homogeneity of nodes and hyperedges. Specifically, VilLain minimizes the entropy of the v-label assignment vectors of each node and hyperedge obtained at propagation steps $1, \cdots, k$:

$$\mathcal{L}_{\text{local}} = \sum_{\ell=1}^{k} \left(\frac{1}{|V|} \sum_{i=1}^{|V|} \mathcal{E}\left(X_i^{(\ell)} \right) + \frac{1}{|E|} \sum_{i=1}^{|E|} \mathcal{E}\left(Y_i^{(\ell)} \right) \right), \tag{4}$$

where $\mathcal{E}(p) = -\sum_i p_i \log p_i$ is the entropy measure of p. That is, we induce structurally close nodes (or hyperedges) to be assigned to the same v-label. Beyond capturing the homogeneity at the hyperedge level, i.e., $\ell = 1$ (Observation 1), the loss function is designed to reproduce the higher-order homogeneity of nodes and hyperedges by minimizing the entropy of v-label assignment vectors at each propagation step $\ell \in [1, k]$ (Observation 2). For training speed, the number of steps k for training can be smaller than k' for inference.

Capturing Global Information. VilLain also considers the global distribution of labels. To this end, we give v-label-level supervision to VilLain so that v-labels are properly distributed over the entire hypergraph. First, since Eq. (4) is trivially minimized when all nodes and hyperedges are assigned to a single v-label, we use the following term to prevent this problem:

$$\mathcal{J}_{\text{cls}} = -\sum_{\ell=1}^{k} \left(\mathcal{E}\left(\mathbf{x}^{(\ell)} \right) + \mathcal{E}\left(\mathbf{y}^{(\ell)} \right) \right) \tag{5}$$

$$\text{where} \quad \mathbf{x}_i^{(\ell)} = \frac{\|X_{:,i}^{(\ell)}\|_1}{\sum_{j=1}^{d} \|X_{:,j}^{(\ell)}\|_1} \quad \text{and} \quad \mathbf{y}_i^{(\ell)} = \frac{\|Y_{:,i}^{(\ell)}\|_1}{\sum_{j=1}^{d} \|Y_{:,j}^{(\ell)}\|_1}.$$

Here, $\mathbf{x}^{(\ell)} = [\mathbf{x}_1^{(\ell)}, \cdots, \mathbf{x}_d^{(\ell)}]$ and $\mathbf{y}^{(\ell)} = [\mathbf{y}_1^{(\ell)}, \cdots, \mathbf{y}_d^{(\ell)}]$ denote the weighted ratios of nodes and hyperedges for each v-label at step ℓ. Note that $X_{:,i}^{(\ell)}$ and $Y_{:,i}^{(\ell)}$, which are the ith columns of $X^{(\ell)}$ and $Y^{(\ell)}$, correspond to the vectors of v-label i for nodes and hyperedges, respectively. That is, we maximize the *entropy of the global distribution* of the v-labels at each step, restraining any single v-label from dominating the entire hypergraph.

In addition, we aim to make v-labels distinctive by making the sets of nodes and hyperedges assigned to each v-label nearly disjoint from those with another v-label. To this end, we minimize the following cross-entropy-based objective:

$$\mathcal{J}_{\text{dst}} = -\sum_{\ell=1}^{k} \sum_{i=1}^{d} \left(\log \bar{\mathbf{x}}_i^{(\ell)} + \log \bar{\mathbf{y}}_i^{(\ell)} \right) \tag{6}$$

$$\text{where} \quad \bar{\mathbf{x}}_i^{(\ell)} = \frac{e^{\mathcal{S}\left(X_{:,i}^{(\ell)}, X_{:,i}^{(\ell)}\right)}}{\sum_{j=1}^{d} e^{\mathcal{S}\left(X_{:,i}^{(\ell)}, X_{:,j}^{(\ell)}\right)}} \quad \text{and} \quad \bar{\mathbf{y}}_i^{(\ell)} = \frac{e^{\mathcal{S}\left(Y_{:,i}^{(\ell)}, Y_{:,i}^{(\ell)}\right)}}{\sum_{j=1}^{d} e^{\mathcal{S}\left(Y_{:,i}^{(\ell)}, Y_{:,j}^{(\ell)}\right)}}.$$

Here, $\bar{\mathbf{x}}_i^{(\ell)}$ and $\bar{\mathbf{x}}_j^{(\ell)}$ indicate the distinctiveness of v-label i at each propgation step ℓ in nodes and hyperedges, respectively, and $\mathcal{S}(\cdot, \cdot)$ measures the cosine similarity of two input vectors. Minimizing Eq. (6) reinforces the *distinctiveness of each v-label* at each step.

Finally, we minimize the global-level loss, defined as the sum of Eq. (5) and Eq. (6), to let v-labels be properly distributed across the entire hypergraph:

$$\mathcal{L}_{\text{global}} = \mathcal{J}_{\text{cls}} + \mathcal{J}_{\text{dst}} \tag{7}$$

Objective Function. To exhibit both the local and global structure-label patterns, VilLain minimizes both objectives, Eq. (4) and (7):

$$\mathcal{L} = \mathcal{L}_{\text{local}} + \mathcal{L}_{\text{global}}.$$

While we can introduce a hyperparameter for balancing $\mathcal{L}_{\text{local}}$ and $\mathcal{L}_{\text{global}}$ (see Appendix D.3), we simply add the two losses since hyperparameter tuning based on external supervision is strictly restricted in our setting.

Note that, by reproducing higher-order label homogeneity, VilLain captures not only structural properties but also potential higher-order structure-label relationships. Consequently, compared to self-supervised methods that exclusively focus on structural aspects, VilLain can learn effective embeddings for less structure-dependent tasks, such as node classification.

Complexity Analysis. We analyze the time and space complexity of VilLain for computing the final embedding Z^*, as well as the computational cost associated with optimizing their losses. Specifically, when the dimension of each subspace is d/D, it takes:

$$O\left(kd \sum_{e \in E} |e| + \frac{kd^2}{D} (|V| + |E|) \right) \text{time} \quad \text{and} \quad O\left(\frac{kd^2}{D} (|V| + |E|) \right) \text{space}$$

Table 1: Summary statistics of eight real-world hypergraphs: the number of nodes $|V|$, the number of hyperedges $|E|$, the size of the hypergraph $\sum_{e\in E}|e|$, the number of edges $|\mathcal{E}|$ in the clique expansion, and the number of ground-truth labels.

Dataset		$\lvert V\rvert$	$\lvert E\rvert$	$\sum_{e\in E}\lvert e\rvert$	$\lvert\mathcal{E}\rvert$	# Labels
Primary (PR)	[59]	242	12,704	30,729	8,317	11
High (HG)	[45]	327	7,818	18,192	5,818	9
Citeseer (CS)	[73]	1,019	819	2,808	3,867	6
Cora (CR)	[73]	1,330	1,503	4,599	4,144	7
Pubmed (PM)	[73]	3,824	7,951	34,605	123,819	3
DBLP (DB)	[73]	36,188	18,924	90,868	425,669	6
Trivago (TV)	[13]	172,738	233,202	726,861	1,095,204	160
Amazon (AZ)	[48]	260,209	31,964	422,076	14,142,811	10

for propagating v-labels (Eq. (2)) and computing losses $\mathcal{L}_{\text{local}}$ (Eq. (4)), $\mathcal{J}_{\text{cls}}$ (Eq. (5)), and $\mathcal{J}_{\text{dst}}$ (Eq. (6)) for $1,\cdots,k$ steps. To generate node embeddings (Eq. (3)), the losses are not necessarily computed, and thus it takes $O(k'd\sum_{e\in E}|e|)$ time and requires $O(k'd(|V|+|E|))$ space. The details can be found in Appendix A. Importantly, introducing multi-v-labels (i.e., $D>1$) leads to the reduction in time and space complexity, thereby indicating an additional advantage of learning v-labels in multiple subspaces.

5.3 Extension to Unobserved Nodes

Heretofore, we described how VilLain learns node embeddings $\mathbf{Z}$ from a static hypergraph. However, in many scenarios, hypergraphs evolve over time (e.g., new members in the group), introducing new nodes and hyperedges to the hypergraph. This motivates us to extend VilLain to generate embeddings also for newly introduced, unobserved nodes and hyperedges. In this subsection, we extend VilLain to embed such unobserved nodes and hyperedges.

Settings. Consider a connected hypergraph $G_{\text{S}}=(V_{\text{S}},E_{\text{S}})$, which is a subset of a connected hypergraph $G=(V,E)$, where $V_{\text{S}}\subseteq V$ and $E_{\text{S}}\subseteq E$. Using the incidence matrix $\mathbf{H}_{\text{S}}\in\{0,1\}^{|V_{\text{S}}|\times|E_{\text{S}}|}$ of G_{S}, VilLain has generated v-labels and embeddings $\mathbf{X}_{\text{S}}^{(0)},\mathbf{Z}_{\text{S}}\in\mathbb{R}^{|V_{\text{S}}|\times d}$, respectively, for the observed nodes V_{S}. Nodes $V\setminus V_S$ and hyperedges $E\setminus E_S$ are introduced after VilLain training.

Embedding Unobserved Nodes. To embed nodes including the unobserved ones $V\setminus V_{\text{S}}$, VilLain propagates learned v-labels $\mathbf{X}_S^{(0)}$ of the observed nodes V_S on hypergraph G containing the unobserved nodes and hyperedges. Specifically, v-label assignment matrices for all nodes $\mathbf{X}^{(\ell)}$ and hyperedges $\mathbf{Y}^{(\ell)}$ at step $\ell\geq 1$ are obtained like in Eq. (2) as follows:

$$\mathbf{Y}^{(\ell)}=\mathbf{D}_E^{-1}\mathbf{H}^T\mathbf{X}^{(\ell-1)}\quad\text{and}\quad\mathbf{X}^{(\ell)}=\mathbf{D}_V^{-1}\mathbf{H}\mathbf{Y}^{(\ell)},$$

where $\mathbf{X}^{(0)}\in\mathbb{R}^{|V|\times d}$ is $\mathbf{X}_S^{(0)}$ with zero-paddings at row indices of the nodes $V\setminus V_{\text{S}}$. Since we assume a connected hypergraph G, there always exists ℓ' such that all nodes V are assigned non-zero v-labels. Then, using Eq. (3), $\mathbf{X}^{(\ell')},\cdots,\mathbf{X}^{(k')}$ are used to generate embeddings $\mathbf{Z}$ for all nodes V, where $k'\geq\ell'$.

6 EXPERIMENTAL RESULTS

In this section, we present experimental results for four downstream tasks utilizing node embeddings. We first assess the accuracy of VilLain by comparing it with the state-of-the-art (hyper)graph representation learning methods (Section 6.2). Then, we demonstrate the effectiveness of each design choice of VilLain (Section 6.3). Lastly, we conduct additional analyses on VilLain (Section 6.4).

6.1 Experimental Settings

In this subsection, we report the experimental settings.

Datasets. We use eight publicly available real-world hypergraphs summarized in Table 1. All datasets are derived from group interactions that arise in real-world scenarios (e.g., coauthorship and co-purchase). For details regarding the preprocessing method and descriptions for each dataset, refer to Appendix C.1.

Baselines. We consider 15 unsupervised and (semi-)supervised graph and hypergraph embedding methods as competitors. Deepwalk [53], Node2vec [23], DGI [63], GRACE [85], GMI [52], Hyper2vec [27], LBSN [75], and TriCL [35] are unsupervised methods, and GCN [34], GAT [62], HGNN [20], HNHN [16], AllSet [10], UniGNN [28], and HyperGCL [66] are (semi-)supervised methods. For graph embedding methods (i.e., GCN, GAT, Deepwalk, Node2vec, DGI, GRACE, and GMI), we use the clique expansion of the hypergraph.[4] For all methods that require node features (i.e., GCN, GAT, DGI, GRACE, GMI, HGNN, HNHN, AllSet, UniGNN, HyperGCL, and TriCL), we use the embeddings obtained by Hyper2vec,[5] which lead to the best performance among three alternatives (see [39] for detailed results).

Implementations. We simply use $k=4$ for VilLain and all its variants and use $k'=10$ for small datasets (s.t., $|V|<10,000$) and $k'=100$ for large datasets (s.t., $|V|\geq 10,000$). As discussed in Section 5.1, to capture diverse structural-label information, we aggregate embeddings obtained with various numbers of v-labels. Specifically, we concatenate embeddings obtained using different numbers of v-labels. For each number $\lceil\frac{d}{D}\rceil\in\{2,3,\cdots,8\}$ of v-labels, we learn D subspaces and then perform PCA to ensure that the final embedding is of the target dimension d. Refer to Appendix C.2 for the detailed settings of other baselines.

6.2 Accuracy of VilLain

To verify the quality of the VilLain's node embeddings, we consider four downstream tasks on hypergraphs: node classification, node retrieval, node clustering, and hyperedge prediction. The embedding dimension of all methods, including VilLain, is fixed to 128. Results including standard deviations are provided in [39].

Node Classification. We perform node classification by randomly and disjointly splitting the dataset into training, validation, and test sets. For training and validation sets, the labels of 20 nodes per class are given for all datasets except for Primary and High, where the labels of 2 nodes are given per class. The remaining nodes are used as the test set. For un- or self-supervised methods including VilLain, we evaluate the accuracy of logistic regression using the embeddings obtained from each method. Table 2 shows the accuracy of all methods in all datasets. VilLain ranks first on average, showing the best performance. We conjecture that v-label propagation inherits rich structural properties and also potential higher-order structure-label relationships, generating high-quality representations of nodes.

[4]The clique expansion is the pairwise graph obtained by replacing each hyperedge with the clique formed by the nodes in the hyperedge.

[5]For Amazon, we use Node2vec since Hyper2vec ran out of time (> 24 hours).

Table 2: VilLain performs best on node classification in terms of accuracy. Each baseline method is designed for either graphs or hypergraphs and for either semi-supervised or unsupervised settings.

Method	DBLP	Trivago	Amazon	Primary	High	Citeseer	Cora	Pubmed	Rank
GCN	67.37 ± 1.45	38.06 ± 1.49	28.73 ± 4.73	75.63 ± 5.08	96.25 ± 2.55	60.64 ± 3.47	72.96 ± 1.82	77.56 ± 2.58	7.00 ± 2.91
GAT	61.74 ± 1.97	51.52 ± 0.68	30.94 ± 2.13	66.79 ± 4.73	90.58 ± 2.76	49.57 ± 2.64	58.09 ± 2.14	73.67 ± 1.78	11.75 ± 3.83
Deepwalk	29.03 ± 1.43	16.85 ± 0.45	25.43 ± 1.72	84.89 ± 3.67	99.31 ± 0.48	45.10 ± 3.18	56.58 ± 1.88	68.58 ± 2.60	11.62 ± 4.71
Node2vec	29.21 ± 1.89	16.88 ± 0.44	25.27 ± 2.36	83.53 ± 3.09	**99.38 ± 0.45**	45.37 ± 3.17	59.15 ± 1.84	69.05 ± 3.00	11.00 ± 4.35
DGI	62.37 ± 3.32	73.46 ± 1.22	31.80 ± 1.45	86.66 ± 4.51	92.49 ± 0.60	61.36 ± 2.91	71.23 ± 2.04	77.51 ± 1.38	7.25 ± 3.59
GRACE	71.86 ± 2.51	OOM	OOM	63.78 ± 5.12	99.03 ± 0.30	61.16 ± 2.78	73.43 ± 1.81	77.70 ± 1.81	5.50 ± 4.75
GMI	64.19 ± 1.63	OOM	OOM	80.10 ± 4.94	96.61 ± 2.63	58.67 ± 2.68	71.31 ± 1.69	75.51 ± 2.77	9.16 ± 1.57
HGNN	66.60 ± 2.18	OOM	OOM	88.28 ± 5.02	92.19 ± 3.84	60.91 ± 2.32	72.90 ± 2.00	76.58 ± 2.86	7.50 ± 3.09
HNHN	63.99 ± 2.21	59.52 ± 1.64	28.99 ± 2.63	91.31 ± 2.47	96.83 ± 1.25	59.02 ± 1.63	68.81 ± 1.26	75.33 ± 1.77	7.50 ± 2.39
AllSet	63.67 ± 1.89	36.58 ± 0.93	21.75 ± 1.67	85.94 ± 3.02	95.70 ± 1.66	56.08 ± 1.95	67.73 ± 1.81	74.11 ± 2.04	10.75 ± 1.08
UniGNN	67.16 ± 2.15	69.98 ± 1.60	33.77 ± 3.22	88.88 ± 3.58	95.12 ± 3.97	59.10 ± 2.76	71.44 ± 1.03	74.37 ± 2.10	7.12 ± 3.09
HyperGCL	58.72 ± 1.54	74.99 ± 1.23	22.86 ± 2.01	74.07 ± 6.06	85.79 ± 8.92	57.54 ± 1.61	74.99 ± 1.33	78.44 ± 3.33	8.87 ± 5.18
Hyper2vec	67.18 ± 1.78	75.82 ± 1.45	OOT	92.52 ± 2.45	96.34 ± 1.34	61.50 ± 2.60	71.79 ± 1.63	77.04 ± 1.51	4.85 ± 2.35
LBSN	22.63 ± 2.20	47.99 ± 0.82	11.56 ± 0.90	86.71 ± 3.71	95.87 ± 2.28	45.43 ± 2.15	59.70 ± 1.31	54.89 ± 2.38	11.87 ± 3.21
TriCL	68.18 ± 1.36	OOM	OOM	92.67 ± 2.50	98.10 ± 1.02	59.17 ± 3.35	72.35 ± 1.53	78.57 ± 1.88	4.16 ± 1.95
VilLain	**77.16 ± 1.26**	**79.43 ± 1.63**	**57.95 ± 2.47**	**93.66 ± 3.93**	99.19 ± 0.41	**61.53 ± 3.17**	**75.03 ± 1.38**	**78.82 ± 1.47**	**1.25 ± 0.66**

Table 3: VilLain performs overall best on hyperedge prediction (in terms of accuracy), node clustering (in terms of normalized mutual information), and node retrieval (in terms of mean average precision).

Method	**Hyperedge Prediction** (Acc.)									**Node Clustering** (NMI)									**Node Retrieval** (MAP)								
	DB	TV	AZ	PR	HG	CS	CR	PM	Rank	DB	TV	AZ	PR	HG	CS	CR	PM	Rank	DB	TV	AZ	PR	HG	CS	CR	PM	Rank
Deepwalk	63.9	61.3	69.4	83.8	85.9	69.6	67.2	65.9	6.25	0.7	16.7	7.8	85.2	**100.0**	14.6	23.9	**34.4**	5.00	21.3	7.5	27.7	81.6	98.7	27.6	29.2	49.0	6.37
Node2vec	64.2	61.4	69.3	83.2	85.4	70.4	66.9	65.8	6.62	0.9	17.0	7.7	83.5	**100.0**	14.5	23.8	32.8	5.87	21.6	7.1	27.8	81.1	98.6	27.3	29.4	49.4	6.50
DGI	**86.1**	83.8	90.8	79.1	84.4	79.2	76.3	80.9	3.75	16.6	44.5	13.0	84.4	73.9	29.1	32.1	31.3	5.62	36.1	37.3	31.1	89.7	97.8	43.8	50.6	61.7	3.25
GRACE	85.4	OOM	OOM	80.3	87.4	77.9	74.5	79.1	4.00	43.0	OOM	OOM	67.6	98.2	33.0	46.0	31.6	5.00	50.2	OOM	OOM	61.4	**99.5**	41.1	54.2	60.9	4.00
GMI	75.6	OOM	OOM	82.4	85.9	74.4	69.4	72.3	5.50	27.8	OOM	OOM	84.1	93.1	25.3	42.6	18.7	6.50	34.6	OOM	OOM	80.0	97.8	35.9	41.6	55.1	6.33
Hyper2vec	71.2	72.4	OOT	76.4	79.6	78.1	71.7	71.5	6.14	43.4	66.3	OOT	**92.5**	99.3	34.3	45.5	33.6	2.27	35.5	43.1	OOT	85.7	90.7	41.2	46.7	55.6	4.85
LBSN	48.7	89.1	63.7	79.4	87.1	74.3	69.6	66.1	5.87	1.1	39.4	2.7	85.5	97.8	12.1	29.0	4.6	6.50	21.0	19.1	29.1	81.3	93.2	30.6	40.1	43.5	6.62
TriCL	77.4	OOM	OOM	**84.0**	87.8	82.0	76.7	80.5	2.33	38.0	OOM	OOM	87.8	98.7	34.4	44.8	33.7	3.00	45.1	OOM	OOM	89.9	97.6	42.4	55.0	61.9	3.16
VilLain	81.6	**95.1**	**94.9**	83.2	**87.8**	**82.1**	**79.0**	**82.8**	**1.50**	**46.6**	**69.4**	**35.2**	85.7	98.7	**34.5**	**50.4**	32.7	**2.25**	**60.2**	**67.2**	**53.6**	**91.3**	99.0	**46.4**	**58.0**	**64.4**	**1.12**

Hyperedge Prediction. The problem of hyperedge prediction is formulated as a binary classification task, predicting whether the given hyperedge is real or fake [30, 51, 78]. Given a set E of real hyperedges, we generate a set E' of fake hyperedges with the same hyperedge size distribution by randomly sampling subsets of nodes. To obtain the embedding of each hyperedge, we apply maxmin pooling [6] to the embeddings of the nodes in it. For more training details on hyperedge prediction, refer to Appendix C.4. As shown in Table 3, VilLain performs the best on average. We conjecture that VilLain, which captures potential structure-label relations, is effective for this task because it indirectly relates to labels due to the high label homogeneity of real hyperedges.

Node Clustering. For the clustering task, we group nodes into the number of unique ground-truth labels, applying k-means to the learned embeddings. Then, we compute the Normalized Mutual Information (NMI) to assess the quality of clustering. As shown in Table 3, VilLain outperforms all baseline methods in terms of average ranks. This indicates that the embeddings learned by VilLain exhibit meaningful semantic similarities in their distribution.

Node Retrieval. The problem of node retrieval aims to search for similar nodes of a given query node, using the learned embeddings. Specifically, we retrieve nodes based on the cosine similarity between their embeddings and the embedding of the query node. Then, we compute the Mean Average Precision (MAP), to measure the retrieval quality. Intuitively, the retrieval is considered to be successful if the nodes of the same class as the query node are highly ranked. For more details regarding the task, refer to Appendix C.3. As shown in Table 3, VilLain outperforms baseline methods, with a large margin. These results imply that v-labels, which are *virtual* and learned without any ground-truth node labels, are useful for finding similar nodes of the same class.

[6]We compute maxmin pooling by: elementwise max pooling - elementwise min pooling. An alternative pooling method is compared in [39].

6.3 Ablation Study

In this subsection, we conduct ablation studies to verify the effectiveness of each component of VilLain by comparing its performance to that of its variants.

Effectiveness of Multi-V-label Learning. To demonstrate the effectiveness of using multiple subspaces, we consider two variants of VilLain: (a) **VilLain-S** learns d v-labels in a single embedding space and (b) **VilLain-M** learns $\lceil d/D \rceil$ v-labels in D subspaces. In Table 4, we compare VilLain with the two variants on the considered tasks. Regarding VilLain-M, we report the average accuracy when $\lceil d/D \rceil = \{2, 3, \cdots, 8\}$. We first observe that VilLain-M consistently outperforms VilLain-S, indicating the effectiveness of the multi-v-label propagation. Additionally, introducing multiple subspaces

Table 4: VilLain outperforms its three variants, VilLain-S, VilLain-M, and VilLain-L, in four downstream tasks, implying that VilLain benefits from (1) propagating v-labels in multiple subspaces, (2) aggregating embeddings from various numbers of v-labels, and (3) reproducing both local and global structure-label patterns for self-supervision.

Method	**Node Classification** (Accuracy)								**Hyperedge Prediction** (Accuracy)								**Node Clustering** (NMI)								**Node Retrieval** (MAP)							
	DB	TV	AZ	PR	HG	CS	CR	PM	DB	TV	AZ	PR	HG	CS	CR	PM	DB	TV	AZ	PR	HG	CS	CR	PM	DB	TV	AZ	PR	HG	CS	CR	PM
VilLain-S	69.5	OOM	OOM	83.1	96.8	59.9	71.0	77.1	79.7	OOM	OOM	79.2	86.4	80.9	75.4	78.9	35.7	OOM	OOM	**88.4**	97.9	21.3	30.3	25.8	45.3	OOM	OOM	65.3	98.4	41.4	47.2	60.9
VilLain-M	74.2	75.1	54.8	91.6	98.6	61.1	73.7	78.5	80.7	95.0	94.7	83.0	87.5	82.4	**79.0**	82.6	43.5	65.4	35.3	87.3	**98.8**	31.9	46.2	**34.7**	49.2	46.7	50.4	86.1	98.7	44.0	53.7	62.1
VilLain-L	76.9	79.3	56.7	64.5	97.6	**61.9**	74.1	78.1	81.4	94.9	94.2	76.6	87.4	**82.9**	78.8	82.1	42.7	66.6	**36.2**	64.0	96.6	**37.1**	43.9	33.0	59.3	66.1	51.6	66.1	97.5	**46.5**	54.9	63.3
VilLain	**77.2**	**79.4**	**58.0**	**93.7**	**99.2**	61.5	**75.0**	**78.8**	**81.6**	**95.1**	**94.9**	**83.2**	**87.8**	82.1	**79.0**	**82.8**	**46.6**	**69.4**	35.2	85.7	98.7	34.5	**50.4**	32.7	**60.2**	**67.2**	**53.6**	**91.3**	**99.0**	46.4	**58.0**	**64.4**

Table 5: VilLain benefits from the long-range propagation of v-labels. Increasing both the number of v-label propagation (k for loss computation and k' for embedding generation) tends to improve the node classification accuracy.

	DB	TV	AZ	PR	HG	CS	CR	PM	Rank
$k = 1$	74.25	78.14	52.16	**94.67**	**99.51**	60.48	74.96	78.21	3.00
$k = 2$	75.76	78.44	55.09	93.43	99.29	60.17	**75.15**	78.97	2.62
$k = 4$	77.16	79.43	57.95	93.66	99.19	61.53	75.03	78.82	2.25
$k = 8$	**78.22**	**80.24**	**59.12**	92.47	98.78	**62.05**	74.24	**79.22**	**2.12**
$k' = 1$	64.71	60.60	48.46	**96.74**	**99.58**	60.62	74.68	77.94	5.62
$k' = 2$	65.29	61.59	49.42	96.36	99.57	60.44	74.70	78.18	5.50
$k' = 4$	66.64	63.25	50.52	96.33	99.39	60.54	74.77	78.29	5.00
$k' = 8$	67.88	65.08	53.22	93.91	99.26	61.29	**75.06**	78.75	4.50
$k' = 16$	70.83	68.28	54.65	94.49	98.86	61.62	74.86	79.12	3.75
$k' = 32$	73.20	72.31	55.80	92.57	98.59	61.96	74.68	79.22	4.00
$k' = 64$	76.47	76.77	56.42	94.21	98.50	62.42	74.25	78.98	4.00
$k' = 128$	**77.62**	**80.63**	**57.46**	88.68	98.09	**63.67**	74.41	**79.37**	**3.50**

enhances the space complexity, as VilLain-M avoids out-of-memory issues in large hypergraphs like Amazon and Trivago, in contrast to VilLain-S, which aligns with our complexity analysis in Section 5. Furthermore, the superior performance of VilLain over VilLain-M implies that aggregating embeddings from various numbers of v-labels captures more informative potential structure-label relations.

Effectiveness of Loss Functions. To examine the effectiveness of the designed loss functions, we consider another variant of VilLain, **VilLain-L**, which only uses the local loss $\mathcal{L}_{\text{local}}$ to learn v-label distributions. As shown in Table 4, VilLain, which jointly optimizes $\mathcal{L}_{\text{local}}$ and $\mathcal{L}_{\text{global}}$ and thus captures both local and global information of the input hypergraph, outperforms VilLain-L, demonstrating the effectiveness of the proposed loss functions. In [39], we analyze when $\mathcal{L}_{\text{global}}$ is particularly beneficial.

Effects of Long-Range V-label Propagation. To examine the effects of the long-range propagation of v-labels, we test how the number of steps k (during loss computation) and k' (during embedding generation) affect the performance of VilLain in node classification. As shown in Table 5, except for Primary and High, which are the smallest datasets, adopting long-range propagation of v-labels is beneficial. In particular, we can see that large datasets (e.g., DBLP, Trivago, and Amazon) benefit from large ks and k's. This tendency holds in other tasks (i.e., hyperedge prediction, node clustering, and node retrieval) as shown in [39]. This implies that the higher-order label homogeneity, which VilLain aims to reproduce, positively affects the performance in downstream tasks.

Table 6: VilLain yields informative embeddings even for unobserved nodes. Fully observed hypergraphs consist of the entire set V of nodes, whereas partially observed hypergraphs only contain the subset $V_S \subseteq V$ of nodes after removing 50% of the hyperedges. Despite a performance decrease compared to its fully observable settings, VilLain outperforms its strongest baseline, TriCL [35] in node classification, even for the set $V \setminus V_S$ of nodes are not observed in VilLain but observed in TriCL during training.

Learning/Node Type			DB	TV	AZ	CS	CR	PM
Fully Observed	VilLain	V_S	78.01	80.03	56.77	62.75	75.43	79.21
		$V \setminus V_S$	66.19	76.07	58.74	57.52	73.46	69.62
Partially Observed	VilLain	V_S	76.45	78.66	53.72	62.49	73.79	77.78
		$V \setminus V_S$	65.86	74.71	55.23	56.42	72.27	69.71
Fully Observed	TriCL	V_S	69.20	OOM	OOM	60.83	72.94	78.99
		$V \setminus V_S$	55.11	OOM	OOM	53.74	70.02	68.60

6.4 Further Analysis of VilLain

In this subsection, we summarize additional experimental results. Here, we consider the node classification task for evaluation, unless otherwise stated.

Scalability of VilLain. We test the scalability of VilLain by measuring its training time. In order to test scalability on larger hypergraphs, we upscale Cora using HyperCL [37] by $2^{\{5.0,5.5,\cdots,8.0\}}$ times. As seen in Figure 4, VilLain scales linearly with the size of the hypergraph and also the number of propagation steps. In addition, the training time decreases with an increased number of subspaces, which is consistent with our time complexity analysis in Section 5.

Performances on Unobserved Nodes. VilLain can generate embeddings for nodes that are not observed during training, as discussed in Section 5.3. Instead of using the original hypergraph $G = (V, E)$, we evaluate how VilLain, after learning embeddings for the subset V_S of nodes from a partial hypergraph $G_S = (V_S, E_S)$, effectively generates node embeddings for both sets V_S and $V \setminus V_S$ of nodes. Indeed, due to the utilization of reduced structural information, it is natural to expect a degraded quality of node embeddings for both V_S and $V \setminus V_S$ sets of nodes in this scenario. This degradation is empirically shown in Table 6 in comparison to the fully-observable setting. However, VilLain outperforms its strongest baseline, TriCL, across six datasets,[7] even when utilizing partial hypergraphs with 50% of hyperedges removed. TriCL, on the other hand, employs complete hypergraphs to learn embeddings for both sets of nodes. This demonstrates the effectiveness of VilLain in

[7]We did not evaluate on Primary and High. Due to their high density, even removing 90% of their hyperedges did not result in any unobserved nodes.

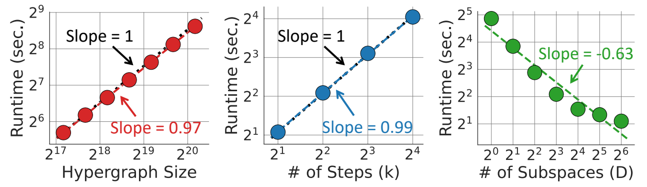

Figure 4: The training time (for 100 epochs) of VilLain is linear in the hypergraph size (i.e., $\sum_{e\in E}|e|$) and the number of steps of v-label propagation (i.e., k). The training time decreases with respect to the number D of subspaces, implying the efficiency of multi-space v-label propagation which is consistent with the complexity analysis in Section 5.

generating informative embeddings for unobserved nodes, as well as its robustness to the removed hyperedges.

Performance on Less Homophilic Hypergraphs. While VilLain is rooted in the insights gained from the observations of higher-order label homogeneity across various real-world hypergraphs (refer to Section 4), it demonstrates a comparable level of performance also in less homophilic hypergraphs. In Figure 5, we generated semi-real hypergraphs by (1) selecting two hyperedges uniformly at random, and (2) interchanging a single node from each. We repeat this process $\{100, 200, \cdots, 1000\}$ and $\{1000, 2000, \cdots, 10000\}$ times in Cora and DBLP, respectively, resulting in hypergraphs with a diverse range of increased hyperedge entropy (i.e., heterophilicity) and thus less homophilic. From the results, we can observe that the node classification accuracies of VilLain in Cora and DBLP degrade with the degree of heterophilicity in the hypergraph. Nonetheless, its performance remains superior to that of the two strongest baselines, Hyper2vec and TriCL, demonstrating its effectiveness in less homophilic hypergraphs as well.

Sensitivity of Multi-V-label Parameters. We analyze how the parameters related to multi-v-label propagation affect the performance of VilLain, specifically the number D of v-label subspaces and the number $\lceil d/D \rceil$ of v-labels in each subspace. As we can see in Figure 6, both the number of subspaces (D) and the number of v-labels in each subspace ($\lceil d/D \rceil$) contribute to the improvement in embedding quality. Empirically, we find that the number of v-labels per subspace has a stronger impact on the performance of VilLain.

Additional Experiments. More experimental results can be found in Appendix D including (1) comparisons with task-specific baselines (Appendix D.1), (2) effects of $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$ (Appendix D.2), and (3) extension of the loss function (Appendix D.3). For additional experimental results, including (1) a detailed analysis of the loss functions (e.g., connections to contrastive losses), (2) improvements from external node features, (3) the usefulness of input features, (4) alternative aggregation methods for embedding generation, and (5) comparisons with graph-modeling-based baselines, refer to [39]. Furthermore, in [39], we develop VilLain$_{\text{B}}$, a space-efficient variant of VilLain that generates binary node embeddings for hypergraphs. Empirical results demonstrate its superior performance compared to baseline methods while requiring only 1/32 of the bits for encoding the node embedding vectors.

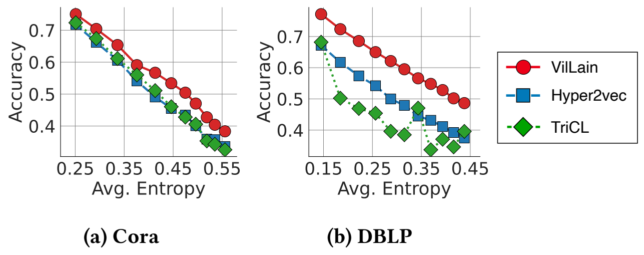

Figure 5: VilLain consistently outperforms Hyper2vec and TriCL in node classification across varying levels of average hyperedge entropy (i.e., heterophilicity).

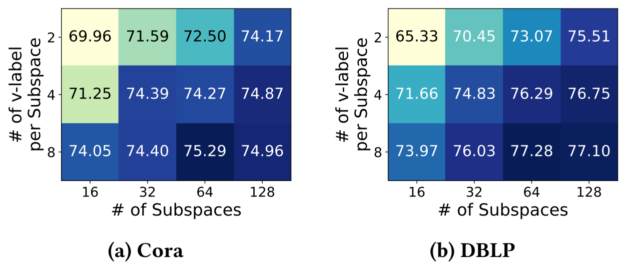

Figure 6: Both the number of subspaces (D) and the number of v-labels in each subspace ($\lceil d/D \rceil$) are positively correlated to the node classification accuracy.

7 CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we propose VilLain for self-supervised node representation learning on hypergraphs. VilLain learns node embeddings that reproduce higher-order label homogeneity in real-world hypergraphs, without requiring external node labels or features. We summarize our contributions as follows:

- **Empirical Findings:** We discover the higher-order homogeneity in real-world hypergraphs, which serves as a guiding principle in the design of VilLain (Section 4).
- **Algorithm Design:** We develop VilLain, a node embedding method for hypergraphs that does not require external information such as labels or features. It produces versatile embeddings that are effective for various tasks (Section 5).
- **Extensive Experiments:** We demonstrate the overall superiority of VilLain over 15 unsupervised and (semi-)supervised competitors on eight datasets in four tasks (Section 6).

While higher-order label homogeneity is observed in a majority of real-world hypergraphs, this may not hold in certain hypergraphs with heterophilic characteristics. Extending VilLain for heterophilic hypergraphs, thus, can be a promising future work.

ACKNOWLEDGEMENTS

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)).

REFERENCES

[1] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. 2021. The surprising power of graph neural networks with random node initialization. In *IJCAI*.

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74, 1 (2002), 47.

[3] Florian Boudin, Ygor Gallina, and Akiko Aa Aizawa. 2020. Keyphrase Generation for Scientific Document Retrieval. In *ACL*.

[4] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *TPAMI* 45, 1 (2022), 657–668.

[5] Derun Cai, Chenxi Sun, Moxian Song, Baofeng Zhang, Shenda Hong, and Hongyan Li. 2022. Hypergraph contrastive learning for electronic health records. In *SDM*.

[6] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. 2023. Graph neural networks for link prediction with subgraph sketching. In *ICLR*.

[7] Abhra Chaudhuri, Ayan Kumar Bhunia, Yi-Zhe Song, and Anjan Dutta. 2023. Data-Free Sketch-Based Image Retrieval. In *CVPR*.

[8] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *ICML*.

[9] Xu Chen, Siheng Chen, Jiangchao Yao, Huangjie Zheng, Ya Zhang, and Ivor W Tsang. 2020. Learning on attribute-missing graphs. *TPAMI* 44, 2 (2020), 740–757.

[10] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. 2021. You are AllSet: A multiset function framework for hypergraph neural networks. In *ICLR*.

[11] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive universal generalized pagerank graph neural network. In *ICLR*.

[12] Uthsav Chitra and Benjamin Raphael. 2019. Random walks on hypergraphs with edge-dependent vertex weights. In *ICML*.

[13] Philip S Chodrow, Nate Veldt, and Austin R Benson. 2021. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances* 7, 28 (2021), eabh1303.

[14] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. 2022. On positional and structural node features for graph neural networks on non-attributed graphs. In *CIKM*.

[15] Hande Dong, Jiawei Chen, Fuli Feng, Xiangnan He, Shuxian Bi, Zhaolin Ding, and Peng Cui. 2021. On the equivalence of decoupled graph convolution network and label propagation. In *WWW*.

[16] Yihe Dong, Will Sawin, and Yoshua Bengio. 2020. HNHN: Hypergraph networks with hyperedge neurons. *arXiv preprint arXiv:2006.12278* (2020).

[17] Dheeru Dua, Casey Graff, et al. 2017. UCI machine learning repository. (2017).

[18] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. 2019. On node features for graph neural networks. *arXiv preprint arXiv:1911.08795* (2019).

[19] Barakeel Fanseu Kamhoua, Lin Zhang, Kaili Ma, James Cheng, Bo Li, and Bo Han. 2021. HyperGraph convolution based attributed HyperGraph clustering. In *CIKM*.

[20] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. In *AAAI*.

[21] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*.

[22] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*.

[23] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *KDD*.

[24] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

[25] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*.

[26] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for pre-training graph neural networks. In *ICLR*.

[27] Jie Huang, Chuan Chen, Fanghua Ye, Jiajing Wu, Zibin Zheng, and Guohui Ling. 2019. Hyper2vec: Biased random walk for hyper-network embedding. In *DASFAA Workshops*.

[28] Jing Huang and Jie Yang. 2021. Unignn: a unified framework for graph and hypergraph neural networks. In *IJCAI*.

[29] Yuchi Huang, Qingshan Liu, and Dimitris Metaxas. 2009. Video object segmentation by hypergraph cut. In *CVPR*.

[30] Hyunjin Hwang, Seungwoo Lee, Chanyoung Park, and Kijung Shin. 2022. Ahp: Learning to negative sample for hyperedge prediction. In *SIGIR*.

[31] TaeHyun Hwang, Ze Tian, Rui Kuangy, and Jean-Pierre Kocher. 2008. Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction. In *ICDM*.

[32] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. Multilevel hypergraph partitioning: Applications in VLSI domain. *VLSI* 7, 1 (1999), 69–79.

[33] Eun-Sol Kim, Woo Young Kang, Kyoung-Woon On, Yu-Jung Heo, and Byoung-Tak Zhang. 2020. Hypergraph attention networks for multimodal learning. In *CVPR*.

[34] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[35] Dongjin Lee and Kijung Shin. 2023. I'm me, we're us, and I'm us: Tri-directional contrastive learning on hypergraphs. In *AAAI*.

[36] Geon Lee, Fanchen Bu, Tina Eliassi-Rad, and Kijung Shin. 2024. A Survey on Hypergraph Mining: Patterns, Tools, and Generators. *arXiv preprint arXiv:2401.08878* (2024).

[37] Geon Lee, Minyoung Choe, and Kijung Shin. 2021. How do hyperedges overlap in real-world hypergraphs?-patterns, measures, and generators. In *WWW*.

[38] Geon Lee, Jihoon Ko, and Kijung Shin. 2020. Hypergraph motifs: concepts, algorithms, and discoveries. *PVLDB* 13, 11 (2020), 2256–2269.

[39] Geon Lee, Soo Yong Lee, and Kijung Shin. 2024. Supplementary Materials. Available online: https://github.com/geon0325/VilLain.

[40] Seongwon Lee, Suhyeon Lee, Hongje Seong, and Euntai Kim. 2023. Revisiting Self-Similarity: Structural Embedding for Image Retrieval. In *CVPR*.

[41] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *NeurIPS*.

[42] Yiran Li, Renchi Yang, and Jieming Shi. 2023. Efficient and Effective Attributed Hypergraph Clustering via K-Nearest Neighbor Augmentation. *PACMMOD* (2023).

[43] Zhonghang Li, Chao Huang, Lianghao Xia, Yong Xu, and Jian Pei. 2022. Spatial-temporal hypergraph self-supervised learning for crime prediction. In *ICDE*.

[44] Xiaorui Liu, Jiayuan Ding, Wei Jin, Han Xu, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Graph neural networks with adaptive residual. In *NeurIPS*.

[45] Rossana Mastrandrea, Julie Fournet, and Alain Barrat. 2015. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PloS one* 10, 9 (2015), e0136497.

[46] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual Review of Sociology* 27, 1 (2001), 415–444.

[47] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[48] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*.

[49] Hoang Nt and Takanori Maehara. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550* (2019).

[50] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*.

[51] Prasanna Patil, Govind Sharma, and M Narasimha Murty. 2020. Negative sampling for hyperlink prediction in networks. In *PAKDD*.

[52] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *WWW*.

[53] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.

[54] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-scale network embedding as sparse matrix factorization. In *WWW*.

[55] Emanuele Rossi, Henry Kenlay, Maria I Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. 2022. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. In *LoG*.

[56] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *AAAI*.

[57] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2021. Random features strengthen graph neural networks. In *SDM*.

[58] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[59] Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella, Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, et al. 2011. High-resolution measurements of face-to-face contact patterns in a primary school. *PloS one* 6, 8 (2011), e23176.

[60] Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P Sadayappan, and Srinivasan Parthasarathy. 2019. Atp: Directed graph embedding with asymmetric transitivity preservation. In *AAAI*.

[61] Shuo Sun, Suzanna Sia, and Kevin Duh. 2020. Clireval: Evaluating machine translation as a cross-lingual information retrieval task. In *ACL*.

[62] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention Networks. In *ICLR*.

[63] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep graph infomax. In *ICLR*.

[64] Changlin Wan, Muhan Zhang, Wei Hao, Sha Cao, Pan Li, and Chi Zhang. 2021. Principled hyperedge prediction with structural spectral features and neural networks. *arXiv preprint arXiv:2106.04292* (2021).
[65] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. 2022. Equivariant and stable positional encoding for more powerful graph neural networks. In *ICLR*.
[66] Tianxin Wei, Yuning You, Tianlong Chen, Yang Shen, Jingrui He, and Zhangyang Wang. 2022. Augmentations in hypergraph contrastive learning: Fabricated and generative. In *NeurIPS*.
[67] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*.
[68] Xiangping Wu, Qingcai Chen, Wei Li, Yulun Xiao, and Baotian Hu. 2020. AdaHGNN: Adaptive hypergraph neural networks for multi-label image classification. In *MM*.
[69] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Huang. 2022. Hypergraph contrastive collaborative filtering. In *SIGIR*.
[70] Lianghao Xia, Chao Huang, and Chuxu Zhang. 2022. Self-supervised hypergraph transformer for recommender systems. In *KDD*.
[71] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-supervised hypergraph convolutional networks for session-based recommendation. In *AAAI*.
[72] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks?. In *ICLR*.
[73] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. HyperGCN: a new method of training graph convolutional networks on hypergraphs. In *NeurIPS*.
[74] Naganand Yadati, Vikram Nitin, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. 2020. Nhp: Neural hypergraph link prediction. In *CIKM*.
[75] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudre-Mauroux. 2019. Revisiting user mobility and social relationships in lbsns: a hypergraph embedding approach. In *WWW*.
[76] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*.
[77] Jaemin Yoo, Hyunsik Jeon, Jinhong Jung, and U Kang. 2022. Accurate node feature estimation with structured variational graph autoencoder. In *KDD*.
[78] Se-eun Yoon, Hyungseok Song, Kijung Shin, and Yung Yi. 2020. How much and when do we need higher-order information in hypergraphs? a case study on hyperedge prediction. In *WWW*.
[79] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. In *AAAI*.
[80] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. 2021. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. In *NeurIPS*.
[81] Junwei Zhang, Min Gao, Junliang Yu, Lei Guo, Jundong Li, and Hongzhi Yin. 2021. Double-scale self-supervised hypergraph learning for group recommendation. In *CIKM*.
[82] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NeurIPS*.
[83] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *NeurIPS*.
[84] Ruochi Zhang, Yuesong Zou, and Jian Ma. 2020. Hyper-SAGNN: a self-attention based graph neural network for hypergraphs. In *ICLR*.
[85] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).
[86] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*.

A DETAILS ON TIME/SPACE COMPLEXITY

In this section, we provide details of the time and space complexity analysis provided in Section 5.

A.1 Details on Time Complexity

Time complexity for v-label propagation. Since we adopt mean-pooling aggregation of $\frac{d}{D}$ v-labels for both nodes and hyperedges in each subspace, it takes $O\left(\frac{d}{D}\sum_{e\in E}|e|\right)$. Thus, it takes:

$$O\left(d\sum_{e\in E}|e|\right)\text{time} \tag{8}$$

for D subspaces, for each step of propagation.

Time complexity for loss computation. In VilLain, there are three losses, $\mathcal{L}_{\text{local}}$, $\mathcal{J}_{\text{cls}}$, and $\mathcal{J}_{\text{dst}}$ that are computed to optimize $\tilde{\mathbf{X}}$.

- For $\mathcal{L}_{\text{local}}$, the entropy of the assignment over $\frac{d}{D}$ v-labels at each node and each hyperedge at each step needs to be computed, and this takes $O\left(\frac{d}{D}(|V|+|E|)\right)$ time for each subspace. Thus, for D subspaces, it takes:
$$O\left(d(|V|+|E|)\right)\quad\text{time} \tag{9}$$
for each propagation step.
- For $\mathcal{J}_{\text{cls}}$, the entropy of the global assignment over $\frac{d}{D}$ v-labels needs to be computed at each step, and this takes $O\left(\frac{d}{D}(|V|+|E|)\right)$ time for each subspace. Thus, for D subspaces, it takes:
$$O\left(d(|V|+|E|)\right)\quad\text{time} \tag{10}$$
for each propagation step.
- For $\mathcal{J}_{\text{dst}}$, $\bar{\mathbf{x}}_1^{(\ell)},\cdots,\bar{\mathbf{x}}_{d/D}^{(\ell)}$ and $\bar{\mathbf{y}}_1^{(\ell)},\cdots,\bar{\mathbf{y}}_{d/D}^{(\ell)}$ are required, and it takes $O\left(\left(\frac{d}{D}\right)^2|V|\right)$ time and $O\left(\left(\frac{d}{D}\right)^2|E|\right)$ time, respectively, to compute them for each subspace. Thus, for D subspaces, it takes:
$$O\left(\frac{d^2}{D}(|V|+|E|)\right)\quad\text{time} \tag{11}$$
for each propagation step.

Thus, from Eq. (8)-(11), the time complexity including (a) v-label propagation and (b) loss computation is:

$$O\left(kd\sum_{e\in E}|e|+\frac{kd^2}{D}(|V|+|E|)\right).$$

Time complexity for embedding generation. To generate node embeddings using Eq. (3), which requires the mean-pooling propagation of v-labels for k' steps, it takes:

$$O\left(k'd(|V|+|E|)\right)\text{ time.}$$

A.2 Details on Space Complexity

Space complexity for v-label propagation. During its v-label propagation, VilLain stores assignment matrices of nodes $\mathbf{X}^{(\ell)}$ and hyperedges $\mathbf{Y}^{(\ell)}$ of $\frac{d}{D}$ v-labels in D subspaces which requires:

$$O\left(kd(|V|+|E|)\right)\quad\text{space}$$

for $\ell=1,\cdots,k$ steps.

Space complexity for loss computation. The losses $\mathcal{L}_{\text{local}}^{(\ell)}$ and $\mathcal{J}_{\text{cls}}^{(\ell)}$ at the ℓth step can be computed directly from $\mathbf{X}^{(\ell)}$ and $\mathbf{Y}^{(\ell)}$, without requiring additional storage space. On the other hand, to compute $\mathcal{J}_{\text{dst}}^{(\ell)}$ at the ℓth step, $\bar{\mathbf{x}}_1^{(\ell)}$ and $\bar{\mathbf{y}}_1^{(\ell)}$ are used, which are computed based on the pairwise cosine similarity between d/D v-labels, requiring $O\left(\frac{d^2}{D}(|V|+|E|)\right)$ space for D subspaces. Thus, the total space required for $\ell=1,\cdots,k$ steps is;

$$O\left(\frac{kd^2}{D}(|V|+|E|)\right).$$

Note that unlike GNN-based methods [16, 20], VilLain does not have any additional learnable parameters in each layer.

Algorithm 1 VilLain's Embedding Generation Scheme

```
Input: (1) Hypergraph $G = (V, E)$, (2) embedding dimension $d$, (3) number
    of subspaces $D$, (4) propagation step $k'$
Output: Node embeddings $\mathbf{Z}$
1:  for each subspace $t = 1, \cdots, D$ in parallel do
2:      $\mathbf{X}^{(0)} \leftarrow \text{Softmax}(\widetilde{\mathbf{X}}) \in [0,1]^{|V| \times \lceil d/D \rceil}$
3:      $\mathbf{Z}^{\langle t \rangle} \leftarrow \mathbf{0}^{|V| \times \lceil d/D \rceil}$
4:      for each step $\ell = 1, \cdots, k'$ do
5:          $\mathbf{Y}^{(\ell)} \leftarrow \mathbf{D}_E^{-1}\mathbf{H}^T\mathbf{X}^{(\ell-1)}$
6:          $\mathbf{X}^{(\ell)} \leftarrow \mathbf{D}_V^{-1}\mathbf{H}^T\mathbf{Y}^{(\ell-1)}$
7:          $\mathbf{Z}^{\langle t \rangle} \leftarrow \mathbf{Z}^{\langle t \rangle} + \mathbf{X}^{(\ell)}$
8:      $\mathbf{Z}^{\langle t \rangle} \leftarrow \mathbf{Z}^{\langle t \rangle} / k'$
9:  $\mathbf{Z} \leftarrow \text{Aggregate}(\mathbf{Z}^{\langle 1 \rangle}, \cdots, \mathbf{Z}^{\langle D \rangle})$
10: return $\mathbf{Z}$
```

Space complexity for embedding generation. To generate node embeddings, $\mathbf{X}^{(\ell)}$ and hyperedge embeddings $\mathbf{Y}^{(\ell)}$ for $\ell = 1, \cdots, k'$ steps are used, and thus $O(k'd(|V| + |E|))$ space is required.

B PSEUDOCODE OF VILLAIN

In Algorithm 1, we present the pseudocode of VilLain, specifically outlining its method for generating embeddings during inference. The embedding for each t^{th} subspace is designed to be produced in parallel. Within each subspace, the probability assignment vector of $\lceil d/D \rceil$ v-labels $\mathbf{X}^{(0)}$ is initialized by applying the Softmax to the learnable embedding $\widetilde{\mathbf{X}}$. Over k' steps, the v-labels are propagated across nodes and hyperedges, where the intermediate node probability assignment vectors are averaged to obtain the subspace's node embedding $\mathbf{Z}^{\langle t \rangle}$. Once D embeddings, $\mathbf{Z}^{\langle 1 \rangle}, \cdots, \mathbf{Z}^{\langle D \rangle}$, from D subspaces, they are concatenated and then PCA is applied, to produce the final embedding $\mathbf{Z}$.

C DETAILS ON EXPERIMENTAL SETTINGS

Here, we provide detailed information on experimental settings.

C.1 Details of Datasets

The statistics of the datasets we used are shown in Table 1.

Preprocessing. For all datasets, we use the largest connected component of the original hypergraph. We process the huge Amazon by remaining nodes that are from the 10 most frequently appeared labels. Then, we randomly sample 1% of the nodes from each label.

Ground-truth labels. Here, we provide how the ground-truth labels of each dataset are assigned. In Primary and High, each node is a person (e.g., student or teacher), and each hyperedge indicates a group interaction among them. If a person is a teacher, then he or she is labeled as a teacher. Otherwise, students are labeled based on the classroom they belong to. In Citeseer, Cora, and Pubmed, which are co-citation hypergraphs, each node is a paper and each hyperedge is a paper that cited the paper. In these hypergraphs, nodes are assigned by their categories. In DBLP, which is a collaboration hypergraph, each node is a paper and each hyperedge is the set of papers written by the same author. Nodes are labeled by their categories. In Trivago, each node is a hotel and each hyperedge is a set of hotels that were clicked in a Web browsing session. Each node is labeled by the location, specifically, the country where the hotel is located. In Amazon, each node is a product, and each hyperedge is a set of products that were co-purchased. Labels of the nodes are assigned by the product categories.

Table 7: Open source links to the baseline source codes.

Method	Github Link
GCN	https://pytorch-geometric.readthedocs.io
GAT	https://pytorch-geometric.readthedocs.io
Deepwalk	https://github.com/benedekrozemberczki/karateclub
Node2vec	https://github.com/benedekrozemberczki/karateclub
DGI	https://github.com/PetarV-/DGI
GRACE	https://github.com/CRIPAC-DIG/GRACE
GMI	https://github.com/zpeng27/GMI
HGNN	https://github.com/iMoonLab/HGNN
HNHN	https://github.com/twistedcubic/HNHN
AllSet	https://github.com/jianhao2016/AllSet
UniGNN	https://github.com/OneForward/UniGNN
HyperGCL	https://github.com/weitianxin/HyperGCL
Hyper2vec	https://github.com/jeffhj/NHNE
TriCL	https://github.com/wooner49/TriCL

C.2 Baselines & Hyperparameters

In this subsection, we discuss the hyperparameters that are used for each method. The implementations we used to run baseline methods are listed in Table 7. Since we consider the unsupervised setting, specifically, without using any labels, the models used for evaluation should be selected without validating on hold-out labeled data. Thus, for unsupervised baseline methods, we either used their default hyperparameter settings or tried to find the settings that generally work well across all datasets. For (semi-)supervised methods, we use the validation set to tune their hyperparameters.

- In VilLain, we fix the number of propagation steps for training to $k = 4$, and for inference, we use $k' = 10$ for small datasets (i.e., Primary, High, Cora, Citeseer, Pubmed) and $k' = 100$ for large datasets (i.e., DBLP, Amazon, and Trivago). The learning rate is fixed to 0.01, and the explained variance ratio of the PCA used in VilLain is fixed to 0.99, throughout the experiments.
- For Deepwalk [53] and Node2vec [23], we use the default hyperparameters. Specifically, we set the number of walks to 10, the length of each walk to 80, the window size to 5, and the learning rate to 0.05. For p and q in Node2vec, we use 1 for both.
- For DGI [63], we use the PReLu for the activation function and set the learning rate to 0.001, as given as default.
- For GRACE [85], we use the ReLU for the activation function, and the number of GCN layers is set to 2. The learning rate and the weight decay rate are set to 0.001 and 0.00001, respectively. Regarding augmentations (e.g., edge drop and feature drop), all rates are set to 0.2. The dimension of the projection head is set to be the same as the hidden dimension.
- For GMI [52], we use the PReLU for the activation function. The learning rate is set to 0.001 without weight decaying. There are three additional hyperparameters α, β, and γ that determine the weights of the local and global mutual information, and they are set to $\alpha = 0.8$, $\beta = 1.0$, and $\gamma = 1.0$, which are default values provided by the authors.
- For HyperGCL [66], we use their default hyperparameters. The number of epochs is set to 500, the augmentation ratio is set to 0.3, the temperature is set to 0.3, and the dropout is set to 0.2.

Table 8: VilLain outperforms specialized baselines, Hyper-SAGNN [84] and NHP [74] in hyperedge prediction.

	DB	TV	AZ	PM	HG	CS	CR	PM	Rank
Hyper-SAGNN	55.7	OOM	OOM	80.2	84.5	76.9	76.8	76.8	2.50
NHP	78.6	73.5	74.3	67.6	72.7	**85.9**	78.8	68.1	2.25
VilLain	**81.6**	**95.1**	**94.9**	**83.2**	**87.8**	82.1	**79.0**	**82.8**	**1.12**

Table 9: VilLain outperforms specialized baselines, GRAC [19] and AHCKA [42] in node clustering.

	DB	TV	AZ	PM	HG	CS	CR	PM	Rank
GRAC	43.9	OOM	22.6	91.6	**99.9**	32.8	46.3	**34.6**	1.85
AHCKA	43.3	OOM	32.0	**93.1**	98.7	32.6	41.8	13.1	2.42
VilLain	**46.6**	**69.4**	**35.2**	85.7	98.7	**34.5**	**50.4**	32.7	**1.50**

- For Hyper2vec [27], the number of walks is set to 10 and the length of each walk is set to 20. The size of the window is 5 and two additional parameters p and q are both set to 1.
- For LBSN [75], the number of negative samples and the learning rate are set to 10 and 0.01, respectively.
- For TriCL [35], we set the number of GCN layers to 1 since it was given as default hyperparameters for most datasets. The learning rate and the weight decaying rate are set to 0.0005 and 0.00001. Regarding the data augmentation, the drop rates for node features and the incidence matrix are both set to 0.4. Three temperature hyperparameters, τ_n, τ_g, and τ_m are all set to 0.5, and two weight hyperparameters w_g and w_m are set to 4 and 1.

C.3 Node Retrieval Protocol

To perform the node retrieval task, we sample $\min(|V|, 1000)$ query nodes from the hypergraph uniformly at random. For each query node, we rank the nodes, excluding the query node, based on the cosine similarity between their learned embeddings and that of the query node. Then, we measure the Mean Average Precision (MAP), which is commonly employed in information retrieval tasks (e.g., computer vision [7, 40] or natural language processing [3, 61]). Here, we define nodes with labels same as that of the query node as the ground-truth. Thus, the MAP yields a higher score when nodes belonging to the same class as the query node are ranked highly.

C.4 Hyperedge Prediction Protocol

To perform the hyperedge prediction task, we first split the original hypergraph $G = (V, E)$ into two sub-hypergraphs $G_{\text{train}} = (V_{\text{train}}, E_{\text{train}})$ and $G_{\text{test}} = (V_{\text{test}}, E_{\text{test}})$ where $E = E_{\text{train}} \cup E_{\text{test}}$ and $E_{\text{train}} \cap E_{\text{test}} = \emptyset$. We also ensure that all nodes are contained in G_{train} (i.e., $V_{\text{train}} = V$) so that embeddings of all nodes in G are learned. Given a train ratio γ, we set the number of hyperedges in G_{train} and G_{test} to be divided based on it, i.e., $|E_{\text{train}}| : |E_{\text{test}}| = \gamma : 1-\gamma$. Specifically, we set $\gamma = 0.80$ for all datasets except for Amazon, which is relatively very sparse, and thus we set $\gamma = 0.95$.

Once we obtain node embeddings of all nodes V, we generate sets of fake hyperedges $E^{\text{fake}}_{\text{train}}$ and $E^{\text{fake}}_{\text{test}}$ as counterparts of true hyperedges E_{train} and E_{test}. Specifically, for each true hyperedge $e \in E_{\text{train}}$ (or E_{test}), we randomly sample $|e|$ nodes from V and create $e' \in E^{\text{fake}}_{\text{train}}$ (or $E^{\text{fake}}_{\text{test}}$). Then, a logistic regression classifier is trained on the $E_{\text{train}} \cup E^{\text{fake}}_{\text{train}}$ and the performance of the hyperedge prediction is evaluated on $E_{\text{test}} \cup E^{\text{fake}}_{\text{test}}$.

Table 10: The two constituent losses, $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$, are effective in capturing the global information of the hypergraph, and thus using both loss terms ($\mathcal{J}_{\text{cls}} + \mathcal{J}_{\text{dst}}$) leads to the best performance in node classification.

	DB	TV	AZ	PM	HG	CS	CR	PM	Rank
$\mathcal{J}_{\text{cls}}$ only	76.42	79.30	57.49	92.52	**99.22**	61.06	74.96	**78.96**	1.75
$\mathcal{J}_{\text{dst}}$ only	76.27	79.26	57.35	92.52	**99.22**	60.85	74.95	78.95	2.50
$\mathcal{J}_{\text{cls}} + \mathcal{J}_{\text{dst}}$	**77.16**	**79.43**	**57.95**	**93.66**	99.19	**61.53**	**75.03**	78.82	**1.50**

Table 11: The optimal value of λ for $\mathcal{L} = \mathcal{L}_{\text{local}} + \lambda\mathcal{L}_{\text{global}}$ varies across datasets w.r.t. node classification performance.

	DB	TV	AZ	PM	HG	CS	CR	PM	Rank
$\lambda = 0$	76.94	79.33	56.73	64.54	97.63	**61.93**	74.13	78.14	3.37
$\lambda = 0.5$	76.64	79.52	57.74	92.02	**99.32**	61.32	74.54	78.49	2.87
$\lambda = 1$	**77.16**	79.43	**57.95**	93.66	99.19	61.53	**75.03**	78.82	**1.50**
$\lambda = 2$	76.73	**79.67**	57.90	**94.87**	99.10	61.37	74.70	**78.86**	2.00

D ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide additional experimental results that are not covered in the main context.

D.1 Task-Specific Baselines

Note that VilLain is designed to generate versatile node embeddings applicable to various downstream tasks, rather than being tailored for a single task. Thus, the used baselines in Table 2 are those that can generate versatile embeddings (e.g., Hyper2Vec and TriCL). Nonetheless, VilLain outperforms additional baselines that are not necessarily versatile. In Table 8, we compare VilLain with Hyper-SAGNN [84] and NHP [74], which are specifically designed for hyperedge prediction. In addition, in Table 9, we compare VilLain with GRAC [19] and AHCKA [42], which are designed for node clustering. In both tasks, VilLain outperforms the baselines, demonstrating its effectiveness and versatility in diverse downstream tasks.

D.2 Effects of $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$ in $\mathcal{L}_{\text{global}}$

To compute $\mathcal{L}_{\text{global}}$ (Eq. (7)), we simply add $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$. We further analyze how the two losses contribute to the final performance by comparing when training VilLain using (1) $\mathcal{J}_{\text{cls}}$ only, (2) $\mathcal{J}_{\text{dst}}$ only, and (3) both $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$. As shown in Table 10, for most datasets, using both $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$ yields superior performance than using either one of them, implying that both $\mathcal{J}_{\text{cls}}$ and $\mathcal{J}_{\text{dst}}$ contribute in learning well-distributed v-labels.

D.3 Extension of the Loss Function

Since we assume a scenario where external information is strictly restricted, even for hyperparameter tuning, we simply add the two losses $\mathcal{L}_{\text{local}}$ and $\mathcal{L}_{\text{global}}$. However, the final loss function can be extended to $\mathcal{L} = \mathcal{L}_{\text{local}} + \lambda\mathcal{L}_{\text{global}}$ where λ is a controllable hyperparameter that balances the two loss terms. Note that $\lambda = 1$ is used for VilLain as default. As shown in Table 11, utilizing $\lambda > 0$ (i.e., integrating $\mathcal{L}_{\text{global}}$) is beneficial across most datasets, particularly in Primary. However, the optimal λ value varies across datasets, and the results suggest that VilLain's performance can be further improved by fine-tuning λ, if the situation allows.