Edge Representation Learning with Hypergraphs

Jaehyeong Jo[1*], **Jinheon Baek**[1*], **Seul Lee**[1*],
Dongki Kim[1], **Minki Kang**[1], **Sung Ju Hwang**[1,2]
KAIST[1], AITRICS[2], South Korea
harryjo97@kaist.ac.kr, jinheon.baek@kaist.ac.kr,
ellenlee7890@gmail.com, cleverki@kaist.ac.kr,
zzxc1133@kaist.ac.kr, sjhwang82@kaist.ac.kr

Abstract

Graph neural networks have recently achieved remarkable success in representing graph-structured data, with rapid progress in both the node embedding and graph pooling methods. Yet, they mostly focus on capturing information from the nodes considering their connectivity, and not much work has been done in representing the *edges*, which are essential components of a graph. However, for tasks such as graph reconstruction and generation, as well as graph classification tasks for which the edges are important for discrimination, accurately representing edges of a given graph is crucial to the success of the graph representation learning. To this end, we propose a novel edge representation learning framework based on *Dual Hypergraph Transformation* (DHT), which transforms the edges of a graph into the nodes of a *hypergraph*. This dual hypergraph construction allows us to apply message-passing techniques for node representations to edges. After obtaining edge representations from the hypergraphs, we then cluster or drop edges to obtain holistic graph-level edge representations. We validate our edge representation learning method with hypergraphs on diverse graph datasets for graph representation and generation performance, on which our method largely outperforms existing graph representation learning methods. Moreover, our edge representation learning and pooling method also largely outperforms state-of-the-art graph pooling methods on graph classification, not only because of its accurate edge representation learning, but also due to its lossless compression of the nodes and removal of irrelevant edges for effective message-passing.[1]

1 Introduction

The recent demand in representing graph-structured data, such as molecular, social, and knowledge graphs, has brought remarkable progress in the *Graph Neural Networks* (GNNs) [61, 54]. Early works on GNNs [32, 20, 56] aim to accurately represent each node to reflect the graph topology, by transforming, propagating, and aggregating information from their neighborhoods based on message-passing schemes [17]. More recent works focus on learning holistic graph-level representations, by proposing graph pooling techniques that condense the node-level representations into a smaller graph or a single vector. While such state-of-the-art node embedding or graph pooling methods have achieved impressive performances on graph-related tasks (e.g., node classification and graph classification), they have largely overlooked the *edges*, which are essential components of a graph.

Most existing GNNs, including ones that consider categorical edge features [45, 17], only implicitly capture the edge information in the learned node/graph representations when updating them. While a

[*] Equal contribution
[1] Code is available at https://github.com/harryjo97/EHGNN

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

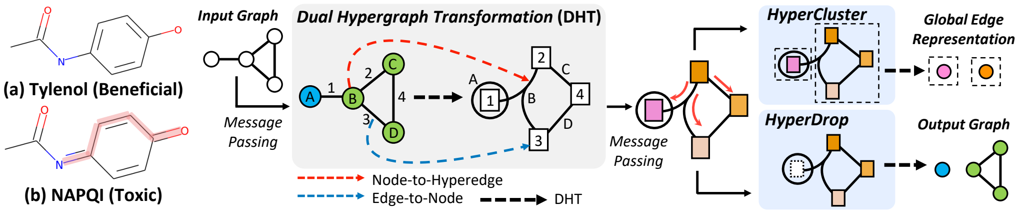


Figure 1: **(Left):** The two molecular graphs[2] have the identical set of nodes, but possess completely different properties due to the difference in edges. **(Right):** An illustration of the proposed edge representation learning framework with two novel edge pooling schemes. The grey box in the center describes the proposed **Dual Hypergraph Transformation**, where the numbers (letters) denote the corresponding edges (nodes) in the graph and nodes (hyperedges) in the hypergraph. The two blue boxes in the right illustrate the proposed edge pooling methods, **HyperCluster** which clusters similar edges, and **HyperDrop** which drops unnecessary edges.

few of them aim to obtain explicit representations for edges [28, 18, 57], they mostly use them only to augment the node-level representations, and thus suboptimally capture the edge information. This is partly because many benchmark tasks for GNN performance evaluation, such as graph classification, do not require the edge information to be accurately preserved. Thus, on this classification task, simple MLPs without any connectivity information can sometimes outperform GNNs [11, 25]. However, for tasks such as graph reconstruction and generation, accurately representing the edges of a graph is crucial to success, as incorrectly reconstructing/generating edges may result in complete failure of the tasks. For example, the two molecules (a) and (b) in Figure 1 have exactly the same set of nodes and are only different in their edges (bond types), but exhibit extremely different properties.

To overcome such limitations of existing GNN methods in edge representation learning, we propose a simple yet effective scheme to represent the edges. The main challenge of handling edges is the absence or suboptimality of the message-passing scheme for edges. We tackle this challenge by representing the edges as nodes in a *hypergraph*, which is a generalization of a graph that can model higher-order interactions among nodes as one hyperedge can connect an arbitrary number of nodes. Specifically, we propose *Dual Hypergraph Transformation* (DHT) to transform edges of the original graph to nodes of a *hypergraph* (Figure 1), and nodes to hyperedges. This hypergraph-based approach is effective since it allows us to apply any off-the-shelf message-passing schemes designed for node-level representation learning, for learning the representation of the edges of a graph.

However, representing each edge well alone is insufficient in obtaining an accurate representation of the entire graph. Thus we propose two novel graph pooling methods for the hypergraph to obtain compact graph-level edge representations, namely *HyperCluster* and *HyperDrop*. Specifically, for obtaining global edge representations for an entire graph, HyperCluster coarsens similar edges into a single edge under the global graph pooling scheme (see HyperCluster in Figure 1). On the other hand, HyperDrop drops unnecessary edges from the original graph by calculating pruning scores on the hypergraph (see HyperDrop in Figure 1). HyperCluster is more useful for graph reconstruction and generation as it does not result in the removal of any edges, while HyperDrop is more useful for classification as it learns to remove edges that are less useful for graph discrimination.

We first experimentally validate the effectiveness of the DHT with *HyperCluster*, on the reconstruction of synthetic and molecular graphs. Our method obtains extremely high performance on these tasks, largely outperforming baselines, which shows its effectiveness in accurately representing the edges. Then, we validate our method on molecular graph generation tasks, and show that it largely outperforms base generation methods, as it allows us to generate molecules with more correct bonds (edges). Further, we validate *HyperDrop* on 10 benchmark datasets for graph classification, on which *HyperDrop* outperforms all hierarchical pooling baselines, with larger gains on social graphs, for which the edge features are important. Our main contributions are summarized as follows:

- We introduce a **novel edge representation learning scheme** using *Dual Hypergraph Transformation*, which exploits the dual hypergraph whose nodes are edges of the original graph, on which we can apply off-the-shelf message-passing schemes designed for node-level representation learning.
- We propose **novel edge pooling methods** for graph-level representation learning, namely *HyperCluster* and *HyperDrop*, to overcome the limitations of existing node-based pooling methods.
- We validate our methods on **graph reconstruction, generation, and classification tasks**, on which they largely outperform existing graph representation learning methods.

[2] We depict only the heavy atoms, as conventional preprocessing of molecular graphs drops hydrogen atoms.

2 Related Work

Graph neural networks Graph neural networks (GNNs) mostly use the message-passing scheme [17] to aggregate features from their neighbors. Particularly, Graph Convolutional Network (GCN) [32] generalizes the convolution operation in the spectral domain of graphs, and updates the representation of each node by applying the shared weights on it and its neighbors' representations. Similarly, GraphSAGE [20] propagates the features of each node's neighbors to itself, based on simple aggregation operations (e.g., mean). Graph Attention Network (GAT) [50] considers the relative importance on neighboring nodes with attention, to update each node's representation as the weighted combination of its neighbors'. Xu et al. [56] show that a simple sum on neighborhood aggregation makes GNNs as powerful as the Weisfeiler-Lehman (WL) test [53], which is effective for distinguishing different graphs. While GNNs have achieved impressive success on graph-related tasks, most of them only focus on learning node-level representations, with less focus on the edges.

Edge-aware graph neural networks Some existing works on GNNs consider edge features while updating the node features [45, 49], however, they only use the edges as auxiliary information and restrict the representation of edges as the discrete features with categorical values. While a few methods [28, 17, 18, 57] explicitly represent edges by introducing edge-level GNN layers, they use the obtained edge features solely for enhancing node features. Also, existing message-passing schemes for nodes are not directly applicable to edge-level layers, as they are differently designed from the node-level layers, which makes it challenging to combine them with graph pooling methods [58] for graph-level representation learning. We overcome these limitations by proposing a dual hypergraph transformation scheme, to obtain a hypergraph whose nodes are edges of the original graph, which allows us to apply any message-passing layers designed for nodes to edges.

Graph transformation Recently, some works [29, 37] propose to transform the original graph into a typical graph structure, to apply graph convolution for learning the edge features. Specifically, they construct a line graph [21], where the nodes of the line graph correspond to the edges of the original graph, and the nodes of the line graph are connected if the corresponding edges of the original graph share the same endpoint. However, the line graph transformation has obvious drawbacks: 1) the transformation is not injective, thus two different graphs may be transformed into the same line graph; 2) the transformation is not scalable; 3) node information in the original graph may be lost during the transformation. Instead of using such a graph structure, we use hypergraphs, which can model higher-order interactions among nodes by grouping multi-node relationships into a single hyperedge [4]. Using the hypergraph duality [44], edges of the original graph are regarded as the nodes of a hypergraph. For example, Lugo-Martinez and Radivojac [36] cast a hyperlink prediction task as an instance of node classification from the dual form of the original hypergraph. On the other hand, Kajino [30] uses the duality to extract useful rules from the hypergraph structures by transforming molecular graphs, for their generation. However, none of the existing works exploit the relation between the original graph and the dual hypergraph for edge representation learning.

Graph pooling Graph pooling methods aim to learn accurate graph-level representation by compressing a graph into a smaller graph or a vector with pooling operations. The simplest pooling approaches are using `mean`, `sum` or `max` over all node representations [1, 56]. However, they treat all nodes equally, and cannot adaptively adjust the size of graphs for downstream tasks. More advanced methods, such as node clustering methods, coarsen the graph by clustering similar nodes based on their embeddings [58, 5], whereas the node pruning methods reduce the number of nodes from the graph by dropping unimportant nodes based on their scores [15, 33]. Ranjan et al. [42] combine both node pruning and clustering approaches, by dropping meaningless clusters after grouping nodes. Baek et al. [2] propose to use attention-based operations for considering relationships between clusters. Note that all of those pooling schemes not only ignore edge representations, but also alter the node set by dropping, clustering, or merging nodes, which result in an inevitable loss of node information.

3 Edge Representation Learning with Hypergraphs

In this section, we first introduce our novel edge representation learning framework with dual hypergraphs, which we refer to as *Edge HyperGraph Neural Network* (EHGNN), and then propose two novel edge pooling schemes for holistic graph-level representation learning: *HyperCluster* and *HyperDrop*. We begin with the descriptions of graph neural networks for node representation learning.

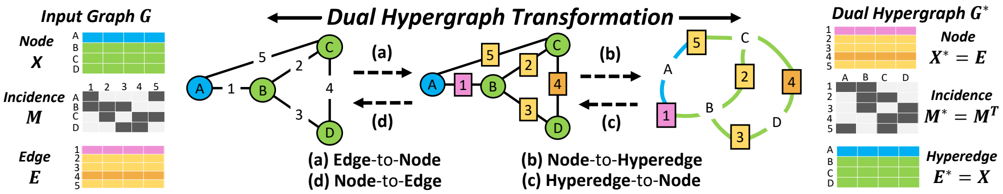


Figure 2: **Dual Hypergraph Transformation.** Illustration of the proposed graph-to-hypergraph transformation.

Graph neural networks A graph G with n nodes and m edges, is defined by its node features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, edge features $\boldsymbol{E} \in \mathbb{R}^{m \times d'}$, and the connectivity among the nodes represented by an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. Here, d and d' are the dimensions of node and edge features, respectively. Then, given a graph, the goal of a *Graph Neural Network* (GNN) is to learn the node-level representation with message-passing between neighboring nodes [17] as follows:

$$\boldsymbol{X}_v^{(l+1)} = \text{UPDATE}\left(\boldsymbol{X}_v^{(l)}, \text{AGGREGATE}\left(\left\{\boldsymbol{X}_u^{(l)} : \forall u \in \mathcal{N}(v; \boldsymbol{A})\right\}\right)\right), \tag{1}$$

where $\boldsymbol{X}^{(l)}$ is the node features at l-th layer, AGGREGATE is the function that aggregates messages from a set of neighboring nodes of the node v, UPDATE is the function that updates the representation of the node v from the aggregated messages, and $\mathcal{N}(v; \boldsymbol{A})$ is the set of neighboring nodes for the node v, obtained from the adjacency matrix $\boldsymbol{A}$. Such message-passing schemes can incorporate the graph topology into each node by updating its representation with the representation of its neighbors.

3.1 Edge representation learning with dual hypergraph transformation

Edge representation learning To reflect the edge information on message-passing, some works on GNNs first obtain the categorical edge features between nodes, and then use them on the AGGREGATE function in equation 1, by adding or multiplying the edge features to the neighboring node's features [17, 45] (see Appendix A.1 for more details). Similarly, few recent works aim to obtain explicit edge representations, but only to use them as auxiliary information to augment the node features, by adding or multiplying edge features to them [18, 57]. Thus, existing works only implicitly capture the edge information in the learned node representations. Although this might be sufficient for most benchmark graph classification tasks, many real-world tasks of graphs (e.g., graph reconstruction and generation) further require the edges to be accurately represented as the information on edges could largely affect the task performance.

Even worse, to define a message-passing function for edge representation learning, existing works propose to additionally create the adjacency matrix for edges, either by defining the edge neighborhood structure [57] or using the line graph transformation [28]. However, these are highly suboptimal as obtaining the adjacency of edges requires $\mathcal{O}(n^2)$ time complexity (see Appendix A.3 for detailed descriptions), as shown in Table 1. This is the main obstacle for directly applying existing message-passing schemes for nodes to edges. To this end, we propose a simple yet effective method to represent the edges of a graph, using a hypergraph.

Table 1: Transformation and message-passing complexities of edge-aware GNNs, line graph, and our EHGNN for the star graph, in which one hub node is connected to n other nodes.

Models	**Complexity**	
	Transformation	**Message-passing**
Edge-aware GNNs	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Line graph	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
EHGNN (Ours)	$\mathcal{O}(n)$	$\mathcal{O}(n)$

Hypergraph A *hypergraph* is a generalization of a graph that can model graph-structured data with higher-order interactions among nodes, wherein a single hyperedge connects an arbitrary number of nodes, unlike in conventional graphs where an edge can only connect two nodes. For example, in Figure 2, the hyperedge B defines the relation among three different nodes. To denote such higher-order relations among arbitrary number of nodes defined by a hyperedge, we use an *incidence matrix* $\boldsymbol{M} \in \{0, 1\}^{n \times m}$, which represents the interaction between n nodes and m hyperedges, instead of using an adjacency matrix $\boldsymbol{A} \in \{0, 1\}^{n \times n}$ that only considers interactions among n nodes. Each entry in the incidence matrix indicates whether the node is incident to the hyperedge. We can formally define a hyperagraph G^* with n nodes and m hyperedges, as a triplet of three components $G^* = (\boldsymbol{X}^*, \boldsymbol{M}^*, \boldsymbol{E}^*)$, where $\boldsymbol{X}^* \in \mathbb{R}^{n \times d}$ is the node features, $\boldsymbol{E}^* \in \mathbb{R}^{m \times d'}$ is the hyperedge features, and $\boldsymbol{M}^* \in \{0, 1\}^{n \times m}$ is the incidence matrix of the hypergraph. We can also represent conventional graphs in the form of a hypergraph, $G = (\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E})$, in which a hyperedge in the incidence matrix $\boldsymbol{M}$ is associated with only two nodes. In the following paragraph, we will describe how to transform the edges of a graph into nodes of a hypergraph, for edge representation learning.

Dual Hypergraph Transformation If we can change the role of the nodes and edges of the graph with a shared connectivity pattern across the nodes and edges, while accurately preserving their information, then we can use any node-based message-passing schemes for learning edges. To achieve this, inspired by the hypergraph duality [4, 44], we propose to transform an edge of the original graph into a node of a hypergraph, and a node of the original graph into a hyperedge of the same hypergraph. We refer to this graph-to-hypergraph transformation as *Dual Hypergraph Transformation* (DHT) (see Figure 2). To be more precise, during the transformation, we interchange the structural role of nodes and edges from the given graph, obtaining the incidence matrix for the new dual hypergraph simply by *transposing* the incidence matrix of the original graph (see the incidence matrix in Figure 2). Along with the structural transformation through the incidence matrix, the DHT naturally interchanges node and edge features across G and G^* (see the feature matrices in Figure 2). Formally, given a triplet representation of a graph, DHT is defined as the following transformation:

$$\boldsymbol{DHT} \; : \; G = \big(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\big) \;\mapsto\; G^* = \big(\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X}\big), \tag{2}$$

where we refer to the transformed G^* as the *dual hypergraph* of the input graph G. Since the dual hypergraph $G^* = (\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X})$ retains all the information of the original graph, we can recover the original graph from the dual hypergraph with the same DHT operation as follows:

$$\boldsymbol{DHT} \; : \; G^* = \big(\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X}\big) \;\mapsto\; G = \big(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\big). \tag{3}$$

This implies that DHT is a *bijective transformation*. DHT is simple to implement, does not incur the loss of any features or topological information of the input graph, and does not require additional memory for feature representations. Moreover, DHT can be sparsely implemented using the edge list, which is the sparse form of the adjacency matrix, by only reshaping the edge list of the original graph into the hyperedge list of the dual hypergraph (see Appendix A.2 for details), which is highly efficient in terms of time and memory. Thanks to DHT, we define the message-passing between edges of the original graph as the message-passing between nodes of its dual hypergraph.

Message-passing on the dual hypergraph for edge representation learning After transforming the original graph into its corresponding dual hypergraph using DHT, we can perform the message-passing between edges of the input graph, by performing the message-passing between nodes of its dual hypergraph $G^* = (\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X})$, which is formally denoted as follows:

$$\boldsymbol{E}_e^{(l+1)} = \text{UPDATE}\left(\boldsymbol{E}_e^{(l)}, \text{AGGREGATE}\left(\left\{\boldsymbol{E}_f^{(l)} : \forall f \in \mathcal{N}(e; \boldsymbol{M}^T)\right\}\right)\right), \tag{4}$$

where $\boldsymbol{E}^{(l)}$ is the node features of G^* at l-th layer, the AGGREGATE function summarizes the neighboring messages of the node e of the dual hypergraph G^*, and the UPDATE function updates the representation of the node e from the aggregated messages. Here $\mathcal{N}(e; \boldsymbol{M}^T)$ is the neighboring node set of the node e in G^*, which we obtain using the incidence matrix $\boldsymbol{M}^T$ of G^*. Furthermore, instead of using the dense incidence matrix, we can sparsely implement the message-passing on the dual hypergraph with the hyperedge list, from which the complexity of message-passing on the dual hypergraph reduces to $\mathcal{O}(m)$, which is equal to the complexity of message-passing between nodes on the original graph (See Appendix A.3 for details). Note that, since the form of equation 4 is the same as the form of equation 1, we can use any graph neural networks which realize the message-passing operation in equation 1, such as GCN [32], GAT [50], GraphSAGE [20], and GIN [56], for equation 4. In other words, to learn the edge representations $\boldsymbol{E}$ of the original graph, we do not require any specially designed layers, but simply need to perform DHT to directly apply existing off-the-shelf message-passing schemes to the transformed dual hypergraph.

To simplify, we summarize the equation 1 as follows: $\boldsymbol{X}^{(l+1)} = \text{GNN}\left(\boldsymbol{X}^{(l)}, \boldsymbol{M}, \boldsymbol{E}^{(l)}\right)$, and the equation 4 as follows: $\boldsymbol{E}^{(l+1)} = \text{GNN}\left(\boldsymbol{E}^{(l)}, \boldsymbol{M}^T, \boldsymbol{X}^{(l)}\right) = \text{EHGNN}\left(\boldsymbol{X}^{(l)}, \boldsymbol{M}, \boldsymbol{E}^{(l)}\right)$, where EHGNN indicates our edge representation learning framework using DHT. After updating the edge features $\boldsymbol{E}^{(L)}$ with EHGNN, $\boldsymbol{E}^{(L)}$ is returned to the original graph by applying DHT on the dual hypergraph G^*. Then, the remaining step is how to make use of these edge-wise representations to accurately represent the edges of the entire graph, which we describe in the next subsection.

3.2 Graph-level edge representation learning with edge pooling

Existing graph pooling methods do not explicitly represent edges. To overcome this limitation, we propose two novel edge pooling schemes: *HyperCluster* and *HyperDrop*.

Graph pooling The goal of graph pooling is to learn a holistic representation of the entire graph. The most straightforward approach for this is to aggregate all the node features with `mean` or `sum` operations [1, 56], but they treat all nodes equally without consideration of which nodes are important for the given task. To tackle this limitation, recent graph pooling methods propose to either cluster and coarsen nodes [58, 5] or drop unnecessary nodes [15, 33]. While they yield improved performances on graph classification tasks, they suffer from an obvious drawback: inevitable loss of both node and edge information. The node information is lost as nodes are dropped and coarsened, and the edge information is lost as edges for the dropped nodes or internal edges for the coarsened nodes are removed. To overcome this limitation, we propose a graph-level edge representation learning scheme.

HyperCluster We first introduce *HyperCluster*, which is a novel edge clustering method to coarsen similar edges into a single edge, for obtaining the global edge representation. Generally, a clustering scheme for nodes of the graph [58, 5] is defined as follows:

$$\boldsymbol{X}^{pool} = \boldsymbol{C}^T \boldsymbol{X'}, \quad \boldsymbol{M}^{pool} = \boldsymbol{C}^T \boldsymbol{M}, \tag{5}$$

where $\boldsymbol{X}^{pool} \in \mathbb{R}^{n_{pool} \times d}$ and $\boldsymbol{M}^{pool} \in \mathbb{R}^{n_{pool} \times m}$ denote the pooled representations, $\boldsymbol{X'} = \text{GNN}\left(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\right) \in \mathbb{R}^{n \times d}$ is the updated node features, and $\boldsymbol{C} \in \mathbb{R}^{n \times n_{pool}}$ is the cluster assignment matrix that is generated from the $\boldsymbol{X'}$. Following this approach, the proposed HyperCluster clusters similar edges into a single edge, by clustering nodes of the dual hypergraph obtained from the original graph via DHT. In other words, we first obtain the node representation of the dual hypergraph $\boldsymbol{E'} = \text{EHGNN}\left(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\right) \in \mathbb{R}^{m \times d'}$, and then cluster the nodes of the dual hypergraph as follows:

$$\boldsymbol{E}^{pool} = \boldsymbol{C}^T \boldsymbol{E'}, \quad (\boldsymbol{M}^{pool})^T = \boldsymbol{C}^T \boldsymbol{M}^T \tag{6}$$

where $\boldsymbol{E}^{pool} \in \mathbb{R}^{m_{pool} \times d'}$ and $\boldsymbol{M}^{pool} \in \mathbb{R}^{n \times m_{pool}}$ denote the pooled edge representation and the incidence matrix of the input graph respectively, and $\boldsymbol{C} \in \mathbb{R}^{m \times m_{pool}}$ is the cluster assignment matrix generated from the input edge features $\boldsymbol{E'}$. Since HyperCluster coarsens the edges rather than dropping them, this edge pooling method is more appropriate for tasks such as graph reconstruction.

HyperDrop We propose another edge pooling scheme, *HyperDrop*, which drops unnecessary edges to identify task-relevant edges, while performing lossless compression of nodes. Conventional node drop methods [15, 33] remove less relevant nodes based on their scores, as follows:

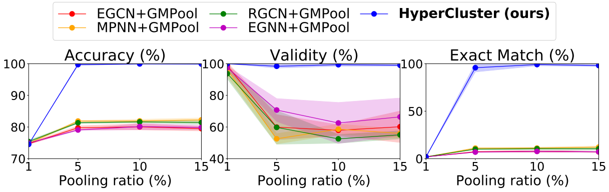
$$\boldsymbol{X}^{pool} = \boldsymbol{X}_{idx}\,, \ \boldsymbol{M}^{pool} = \boldsymbol{M}_{idx}\,; \quad idx = \text{top}_k(\text{score}(\boldsymbol{X})), \tag{7}$$

where idx is the row-wise (i.e., node-wise) indexing vector, $\text{score}(\cdot)$ computes the score of each node with learnable parameters, and $\text{top}_k(\cdot)$ selects the top k elements in terms of the score. However, this approach results in the inevitable loss of node information, as it drops nodes. Thus, we propose to coarsen the graph by dropping edges instead of nodes, exploiting edge representations obtained from our EHGNN. *HyperDrop* selects the top-ranked edges of the original graph, by selecting the top-ranked nodes of the dual hypergraph. The pooling procedure for HyperDrop is as follows:

$$\boldsymbol{E}^{pool} = \boldsymbol{E}_{idx}\,, \ (\boldsymbol{M}^{pool})^T = (\boldsymbol{M}^T)_{idx}\,; \quad idx = \text{top}_k\left(\text{score}(\boldsymbol{E})\right). \tag{8}$$

Then, we can obtain the pooled graph $G^{pool} = (\boldsymbol{X}, \boldsymbol{M}^{pool}, \boldsymbol{E}^{pool})$ by applying DHT to the pooled dual hypergraph. HyperDrop is most suitable for graph classification tasks, as it identifies discriminative edges for the given task. Since HyperDrop preserves the nodes intact, it can also be used for node-level classification tasks, which is impossible with exiting graph pooling methods that modify nodes. In another point of view, the proposed HyperDrop can be further considered as a learnable graph rewiring operation, which optimizes the graph for the given task by deciding whether to drop or maintain the nodes. Finally, a notable advantage of such HyperDrop is that it alleviates the over-smoothing problem in deep GNNs [35] (i.e., the features of all nodes converge to the same values when stacking a large number of GNN layers). As HyperDrop *learns* to remove unnecessary edges, the message-passing only happens across relevant nodes, which alleviates over-smoothing.

Figure 3: **Edge reconstruction results** on the ZINC molecule dataset by varying the pooling ratio. Solid lines denote the mean, and shaded areas denote the standard deviation of 5 runs.



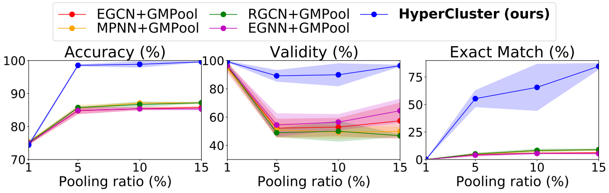
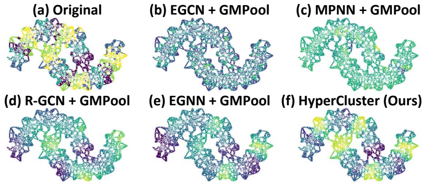


Figure 5: **Graph reconstruction results** on the ZINC molecule dataset by varying the pooling ratio. Solid lines denote the mean, and shaded areas denote the standard deviation of 5 runs.

Figure 4: **Edge reconstruction results** of the synthetic two-moon graph. The edge features are represented by colors.



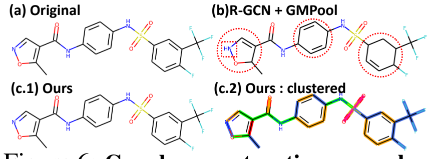


Figure 6: **Graph reconstruction examples.** Red dashed circles and squares indicate the incorrectly predicted edges and nodes, respectively. (c.2) shows the assigned clusters of edges as colors using our method.

4 Experiments

We experimentally validate the effectiveness of EHGNN coupled with either HyperCluster or HyperDrop on four different tasks: graph reconstruction, generation, classification, and node classification.

4.1 Graph reconstruction

Accurately reconstructing the edge features is crucial for graph reconstruction tasks, and thus we validate the efficacy of our method on graph reconstruction tasks first.

Experimental setup We first validate our EHGNN with HyperCluster on the *edge reconstruction* tasks, where the goal is to reconstruct the edge features from their compressed representations. Then, we evaluate our method on the graph reconstruction tasks to validate the effectiveness of ours in holistic graph-level learning. We start with edge reconstruction of a synthetic two-moon graph, where node features (coordinates) are fixed and edge features are colors. For edge and graph reconstruction of real-world graphs, we use the ZINC dataset [27] that consists of 12K molecular graphs [9], where node features are atom types and edge features are bond types. We use accuracy, validity, and exact match as evaluation metrics. For more details, please see Appendix C.1.

Implementation details and baselines We compare the proposed EHGNN framework against edge-aware GNNs, namely EGCN [22], MPNN [17], R-GCN [45], and EGNN [18], which use the edge features as auxiliary information for updating node features. We further combine them with an existing graph pooling method, namely GMPool [2], to obtain a graph-level edge representation for a given graph. In contrast, for our method, we first obtain edge representations with EHGNN, using GCN [32] as the message-passing function, and then coarsen the edge-wise representations using HyperCluster, whose cluster assignment matrices are obtained using GMPool [2]. For node reconstruction, we set message-passing to GCN and graph pooling to GMPool [2] for all models. We provide further details of the baselines and our model in Appendix C.1.

Edge reconstruction results Figure 4 shows the original two-moon graph and edge-reconstructed graphs, where edge features are represented as colors, exhibiting clustered patterns. The baselines fail to reconstruct the edge colors, since they implicitly learn edge representations by using edge features as auxiliary information to update nodes, hence mixing the colors of the neighboring edges. On the other hand, our method distinguishes each edge cluster, which shows that our method can capture meaningful edge information by clustering similar edges. Moreover, as shown in Figure 3, our model obtains significantly higher performance over all baselines on the edge reconstruction task of molecular graphs, in all evaluation metrics. The performance gain of our method over baselines is notably large in exact match, which demonstrates that explicit learning of edge representation is essential for the accurate encoding of the edge information.

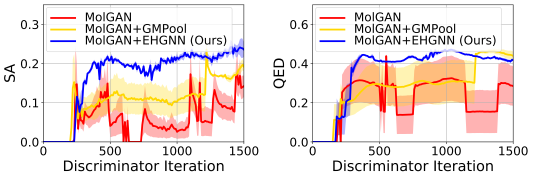


Figure 7: **Graph generation results on MolGAN.** Solid lines denote the mean, and shaded areas denote the standard deviation of 3 different runs.

Dataset	Metrics	MARS [55]	MARS + EHGNN (Ours)
ZINC15	Success Rate	59.53 ± 2.11	**64.30** ± 1.54
	QED (≥ 0.67)	95.71 ± 0.09	**96.36** ± 0.49
	GSK3β (≥ 0.6)	86.52 ± 1.67	**90.63** ± 2.57
	JNK3 (≥ 0.6)	71.52 ± 4.15	**73.60** ± 1.29

Table 2: **Graph generation results on MARS.** The results are the mean and standard deviation of 3 different runs. Best performance and its comparable results ($p > 0.05$) from the t-test are highlighted in bold.

Graph reconstruction results To verify the effectiveness of learning accurate edge representations for reconstructing both the node and edge features, we now validate our method on the molecular graphs in Figure 5. Combining our edge representation learning method (EHGNN + HyperCluster) with the existing node representation learning method (GCN + GMPool) yields incomparably high reconstruction performance compared to the baselines in exact match, which demonstrates that learning accurate edge representation, as well as node representation, is crucial to the success of the graph representation learning methods on graph reconstruction.

Qualitative analysis We visualize the original and reconstructed molecular graphs in Figure 6. As shown in Figure 6 (b), the baseline cannot reconstruct the ring structures of the molecule, whereas our method perfectly reconstructs the rings as well as the atom types. The generated edge clusters in Figure 6 (c.2) further show that our method captures the detailed substructures of the molecule, as we can see the cluster patterns of hexagonal and pentagonal rings. More reconstruction examples of molecular graphs are shown in Figure 13.

Graph compression To validate the effectiveness of our method in large and dense graph compression, we further apply EHGNN with HyperCluster to the Erdos-Renyi random graph [10] having six discrete edge features, where the number of nodes is fixed to 10^3 while the number of edges increases from 10^3 to 10^4. In Figure 8, we report the relative memory size of the compressed graph after pooling the features, against the size of the original graph. We compare our method which compresses both the nodes and edges, against the node pooling method, namely GMT [2]. As the number of edges increases, we observe that compressing only the node features is insufficient for obtaining compact representations, whereas our method is able to obtain highly compact but accurate representation which can be assured from the sufficiently high edge reconstruction accuracy. We believe that our proposed framework can not only learn accurate representations of nodes and edges, but also effectively compress their features, especially for large-scale real-world graphs, such as social networks or protein-protein interaction (PPI) graphs.

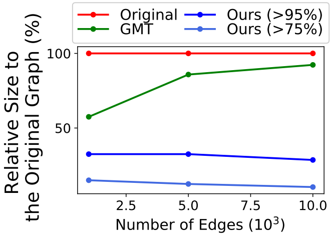


Figure 8: **Graph compression results.** For ours, we report the relative size to the original graph where the edge reconstruction accuracy is higher than 95% and 75%, respectively.

4.2 Graph generation

As shown in Figure 1 (left), graph generation depends heavily on the edge representations, as the model may generate incorrect graphs (e.g., toxic chemicals rather than drugs) if the edge information is inaccurate. Thus, we further validate our EHGNN on the graph generation tasks.

Experimental setup We directly forward the edge representation from the EHGNN to molecule generation networks, namely MolGAN [6] and MArkov moleculaR Sampling (MARS) [55]. MolGAN uses the Generative Adversarial Network (GAN) [19], to generate the molecular graph by balancing weights between its generator and discriminator. MolGAN uses R-GCN [45] for node-level message-passing, whereas, for ours, we first obtain the edge representations using EHGNN, and use them with mean pooling in the graph encoder. For evaluation metrics, we use the Synthetic Accessibility (SA) and Druglikeness (QED) scores. We further apply EHGNN to MARS [55] that generates the molecule by sequentially adding or deleting its fragment, with MCMC sampling. While the original model uses MPNN [17] to implicitly obtain edge representations for adding and deleting actions, we use EHGNN to explicitly learn edge representations. We train models to maximize four

Table 3: **Graph classification results**. The results are the mean and standard deviation over 10 different runs. Best performance and its comparable results ($p > 0.05$) from the t-test are highlighted in bold. Hyphen (-) denotes out-of-resources that take more than 10 days. The results for the baselines are taken from Baek et al. [2].

		TU : Biochemical			**TU : Social**			**OGB : Molecule**				**Average**
		D&D	**PROTEINS**	**MUTAG**	**IMDB-B**	**IMDB-M**	**COLLAB**	**HIV**	**Tox21**	**ToxCast**	**BBBP**	
# graphs		1178	1113	188	1000	1500	5000	41127	7831	8576	2039	
# classes		2	2	2	2	3	3	2	12	617	2	
Avg # nodes		284.32	39.06	17.93	19.77	13.00	74.49	25.51	18.57	18.78	24.06	
Avg # edges		715.66	72.82	19.79	96.53	65.94	2457.78	27.47	19.27	19.26	25.95	
Set	DeepSet	77.39 ± 0.67	68.95 ± 0.92	72.56 ± 1.09	72.42 ± 0.36	50.24 ± 0.32	75.27 ± 0.21	71.20 ± 1.26	72.25 ± 0.23	59.44 ± 0.39	63.64 ± 0.62	68.34
Naive GNN	GCN	72.05 ± 0.55	73.24 ± 0.73	69.50 ± 1.78	73.26 ± 0.46	50.39 ± 0.41	80.59 ± 0.27	76.81 ± 1.01	75.04 ± 0.80	60.63 ± 0.51	65.47 ± 1.73	69.70
	GIN	70.79 ± 1.17	71.46 ± 1.66	81.39 ± 1.53	72.78 ± 0.86	48.13 ± 1.36	78.19 ± 0.63	75.95 ± 1.35	73.27 ± 0.84	60.83 ± 0.46	**67.65** ± 3.00	70.04
Global	SortPool	75.58 ± 0.72	73.17 ± 0.88	71.94 ± 3.55	72.12 ± 1.12	48.18 ± 0.83	77.87 ± 0.47	71.82 ± 1.63	69.54 ± 0.75	58.69 ± 1.71	65.98 ± 1.70	68.49
	GMT	**78.72** ± 0.59	**75.09** ± 0.59	83.44 ± 1.33	73.48 ± 0.76	50.66 ± 0.82	80.74 ± 0.54	**77.56** ± 1.25	**77.30** ± 0.59	**65.44** ± 0.58	**68.31** ± 1.62	73.07
Hierarchical	DiffPool	77.56 ± 0.41	73.03 ± 1.00	79.22 ± 1.02	73.14 ± 0.70	**51.31** ± 0.72	78.68 ± 0.43	75.64 ± 1.86	74.88 ± 0.81	62.28 ± 0.56	**68.25** ± 0.96	71.40
	SAGPool	74.72 ± 0.82	71.56 ± 1.49	73.67 ± 4.28	72.55 ± 1.28	50.23 ± 0.44	78.03 ± 0.31	71.44 ± 1.67	69.81 ± 1.75	58.91 ± 0.80	63.94 ± 2.59	68.49
	TopKPool	73.63 ± 0.55	70.48 ± 1.01	67.61 ± 3.36	71.58 ± 0.95	48.59 ± 0.72	77.58 ± 0.85	72.27 ± 0.91	69.39 ± 2.02	58.42 ± 0.91	65.19 ± 2.30	67.47
	MinCutPool	**78.22** ± 0.54	74.72 ± 0.48	79.17 ± 1.64	72.65 ± 0.75	51.04 ± 0.70	80.87 ± 0.34	75.37 ± 2.05	75.11 ± 0.69	62.48 ± 1.33	65.97 ± 1.13	71.56
	ASAP	76.58 ± 1.04	73.92 ± 0.63	77.83 ± 1.49	72.81 ± 0.50	50.78 ± 0.75	78.64 ± 0.50	72.86 ± 1.40	72.24 ± 1.66	58.09 ± 1.62	63.50 ± 2.47	69.73
	EdgePool	75.85 ± 0.58	**75.12** ± 0.76	74.17 ± 1.82	72.46 ± 0.74	50.79 ± 0.59	-	72.66 ± 1.70	73.77 ± 0.68	60.70 ± 0.92	67.18 ± 1.97	-
	HaarPool	-	-	66.11 ± 1.50	73.29 ± 0.34	49.98 ± 0.57	-	-	-	-	66.11 ± 0.82	-
Ours	HyperDrop	**78.74** ± 0.68	**75.30** ± 0.45	84.00 ± 0.69	73.96 ± 0.41	**51.68** ± 0.41	**81.29** ± 0.25	76.79 ± 0.86	76.95 ± 0.32	64.21 ± 0.70	**69.04** ± 0.86	**73.20**
	HyperDrop + GMT	**78.39** ± 0.33	**75.39** ± 0.26	**85.72** ± 0.61	**74.45** ± 0.61	**51.45** ± 0.28	80.59 ± 0.33	**77.84** ± 0.37	**77.58** ± 0.43	**65.15** ± 0.65	**69.16** ± 1.04	**73.57**

molecule properties: inhibition scores against two proteins, namely GSK3β and JNK3 (biological); QED and SA scores (non-biological). Then we report the success rate at which the molecule satisfies all the properties. For more details, please see Appendix C.2.

MolGAN results Figure 7 shows the SA and QED scores of the generated molecules, of the MolGAN architecture with different encoders. Our EHGNN framework obtains significantly improved generation performance, over the original MolGAN which uses the R-GCN encoder and the MolGAN with GMPool, a state-of-the-art global node pooling encoder. This is because EHGNN learns explicit edge representation which enhances the ability of the discriminator to distinguish between real and generated graphs. The improvement in the discriminator also leads to notably more stable results compared to the baselines, which show a large variance in the quality of the generated molecules.

MARS results To perform correct editing actions to generate graphs with MARS, we need accurate edge representations, as edges determine the structure of the generated molecule. Table 2 shows that using our EHGNN achieves significantly higher generation performance over original MARS, which uses edges as auxiliary information only to enhance node representations. Notably, performance gain on the GSK3β inhibition score for which structural binding is important, suggests that accurate learning of edges leads to generating more effective molecules that interact with the target protein.

4.3 Graph and node classification

Now, we validate the performance of our EHGNN with HyperDrop on classification tasks. Our approach is effective for the classification of graphs with or without edge features, since it allows lossless compression of nodes, and drops edges to allow message-passing only across relevant nodes.

Experimental setup Following the experimental setting of Baek et al. [2], we use the GCN as the node-level message-passing layers for all models, and compare our edge pooling method against the existing graph pooling methods. For this experiment, our HyperDrop uses SAGPool [33] on the hypergraph, which is a node drop pooling method based on self-attention. We use 6 datasets from the TU datasets [39] including three from the biochemical domains (i.e., DD, PROTEINS, MUTAG) and the remaining half from the social domains (i.e., IMDB-BINARY, IMDB-MULTI, COLLAB). Also, we further use the 4 molecule datasets (i.e., HIV, Tox21, ToxCast, BBBP) from the recently released OGB datasets [22]. We evaluate the accuracy of each model with 10-fold cross validation [60] on the TU datasets, and use ROC-AUC as the evaluation metric for the OGB datasets. For both datasets, we follow the standard experimental settings, from the feature extraction to the dataset splitting. We provide additional details of the experiments in Appendix C.3.

Baselines We compare our EHGNN with HyperDrop, against the set encoding (DeepSet [59]), GNNs with naive pooling baselines (GCN and GIN [32, 56]), and state-of-the-art hierarchical pooling methods (DiffPool [58], SAGPool [33], TopKPool [15], MinCutPool [5], ASAP [42], EdgePool [8], and HaarPool [52]) that drop or coarsen node representations. We also additionally compare or combine the state-of-the-art global node pooling methods (SortPool [60], GMT [2]) with our model, for example, HyperDrop + GMT. For more details, see Appendix C.3.

Model	MUTAG	PROTEINS	Tox21
HyperDrop	84.00 ± 0.69	**75.30** ± 0.45	**76.95** ± 0.32
HyperCluster	**84.50** ± 1.50	72.76 ± 1.12	76.68 ± 0.56
w/ RandDrop	83.06 ± 1.15	74.92 ± 0.51	76.39 ± 0.47
w/o HyperDrop	83.06 ± 1.20	75.08 ± 0.37	76.60 ± 0.45
w/o EHGNN	69.50 ± 1.78	73.24 ± 0.73	75.04 ± 0.80

Table 4: **Ablation study** of HyperDrop on the MUTAG, PROTEINS, and Tox21 datasets for classification.

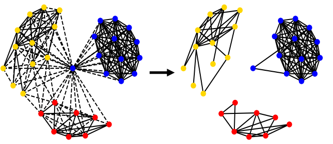


Figure 9: **Edge pooling results** on the COLLAB dataset. Colors denote connected components.

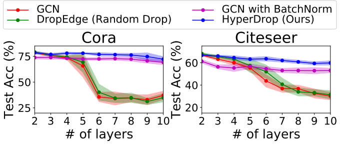


Figure 10: **Node classification results.** Lines denote means over 10 runs and shades denote variances.

Classification results Table 3 shows that the proposed EHGNN with HyperDrop significantly outperforms all hierarchical pooling baselines. This is because HyperDrop not only retains nodes by removing edges that are less useful for graph discrimination, but also explicitly uses the edge representations for graph classification. Since HyperDrop does not remove any nodes on the graph, it can be jointly used with any node pooling methods, and thus, we pair HyperDrop with GMT. This model largely outperforms GMT, obtaining the best performance on most of the datasets, which demonstrates that accurate learning of both the nodes and edges is important for classifying graphs. We further visualize the edge pooling process of HyperDrop in Figure 9, which shows that our method accurately captures the substructures of the entire graph, which leads to dividing the large graph into several connected components, thus adjusting the graph topology for more effective message-passing. We provide more visual examples of edge drop procedures in Appendix D.3.

Ablation study To see how much each component contributes to the performance gain, we conduct an ablation study on EHGNN with HyperDrop. Table 4 shows that, compared with a model that only uses node features (i.e., w/o EHGNN), learning explicit edge representations significantly improves performance. Our model EHGNN with HyperCluster, or without HyperDrop, or the model with random edge drop obtains decent performance, but substantially underperforms HyperDrop.

Over-smoothing with deep GNNs Lastly, we demonstrate that our EHGNN with HyperDrop alleviates the over-smoothing problem of deep GNNs on semi-supervised node classification tasks, which is not possible for the existing node-based pooling methods. We follow the settings of existing works [32, 50, 13] and provide the experimental details in Appendix C.4. As shown in Figure 10, HyperDrop retains the accuracy as the number of layers increases, whereas the naive GCN or random drop [43] results in largely degraded performance, since HyperDrop identifies and preserves the task-relevant edges while the sampling-based methods randomly drop the edges. Further, our method outperforms BatchNorm which alleviates over-smoothing by yielding differently normalized feature distribution at each batch. This is because HyperDrop splits the given graph into smaller subgraphs that capture meaningful message-passing substructures as shown in Figure 9.

5 Conclusion

We tackled the problem of accurately representing the edges of a graph, which has been relatively overlooked over node representation learning. To this end, we proposed a novel edge representation learning framework using *Dual Hypergraph Transformation* (DHT), which transforms the edges of the original graph into nodes on a hypergraph. This allows us to apply a message-passing scheme for node representation learning, for edge representation learning. Further, we proposed two edge pooling methods to obtain a holistic edge representation for a given graph, where one clusters similar edges into a single edge for graph reconstruction and the other drops unnecessary edges for graph classification. We validated our edge representation learning framework on graph reconstruction, generation, and classification tasks, showing its effectiveness over relevant baselines.

6 Acknowledgements and Disclosure of Funding

We thank the anonymous reviewers for their constructive comments and suggestions. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program (KAIST), and No.2021-0-02068, Artificial Intelligence Innovation Hub), and the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

References

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1993–2001, 2016.

[2] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[3] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Claude Berge. Graphs and hypergraphs. 1973.

[5] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. *arXiv preprint*, arXiv:1907.00481, 2019.

[6] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint*, arXiv:1805.11973, 2018.

[7] Michaël Defferrard, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. Pygsp: Graph signal processing in python, 10 2017. URL https://doi.org/10.5281/zenodo.1003158.

[8] Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

[9] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint*, arXiv:2003.00982, 2020.

[10] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[11] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[12] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[13] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[14] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[15] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 2019.

[16] Anna Gaulton, Anne Hersey, Michał Nowotka, A Patricia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2017.

[17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[18] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 9211–9219. Computer Vision Foundation / IEEE, 2019.

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint*, arXiv:1406.2661, 2014.

[20] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[21] Frank Harary and R. Z. Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9:161–168, 1960.

[22] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint*, arXiv:2005.00687, 2020.

[23] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[24] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Vijay S. Pande Percy Liang, and Jure Leskovec. Strategies for pre-training graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[25] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[27] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A free tool to discover chemistry for biology. *J. Chem. Inf. Model.*, 52(7):1757–1768, 2012.

[28] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2656–2662. ijcai.org, 2019.

[29] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2656–2662. ijcai.org, 2019.

[30] Hiroshi Kajino. Molecular hypergraph grammar with its application to molecular optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3183–3191. PMLR, 2019.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, arXiv:1412.6980, 2014.

[32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[33] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 2019.

[34] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 177–187. ACM, 2005.

[35] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3538–3545. AAAI Press, 2018.

[36] Jose Lugo-Martinez and Predrag Radivojac. Classification in biological networks with hypergraphlet kernels. *arXiv preprint*, arXiv:1703.04823, 2017.

[37] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M. Bronstein. Dual-primal graph convolutional networks. *arXiv preprint*, arXiv:1806.00770, 2018.

[38] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint*, arXiv:2007.08663, 2020.

[39] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2014–2023. JMLR.org, 2016.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[42] Ekagra Ranjan, Soumya Sanyal, and Partha P. Talukdar. ASAP: adaptive structure aware pooling for learning hierarchical graph representations. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5470–5477. AAAI Press, 2020.

[43] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[44] Edward Scheinerman and Daniel Ullman. *Fractional graph theory: a rational approach to the theory of graphs*. Courier Coporation, 2011.

[45] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[46] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

[47] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 29–38. IEEE Computer Society, 2017.

[48] Teague Sterling and John Irwin. Zinc 15 - ligand discovery for everyone. *Journal of chemical information and modeling*, 55, 10 2015. doi: 10.1021/acs.jcim.5b00559.

[49] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[50] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[51] Yu Guang Wang and Xiaosheng Zhuang. Tight framelets on graphs for multiscale data analysis. In Dimitri Van De Ville, Manos Papadakis, and Yue M. Lu, editors, *Wavelets and Sparsity XVIII*, volume 11138, pages 100 – 111. International Society for Optics and Photonics, SPIE, 2019.

[52] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montúfar, Xiaosheng Zhuang, and Yanan Fan. Haarpooling: Graph pooling with compressive haar basis. *arXiv preprint arXiv:1909.11580*, 2019.

[53] B. Yu. Weisfeiler and A. A. Leman. Reduction of a graph to a canonical form and an algebra arising during this reduction. 1968.

[54] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[55] Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. Mars: Markov molecular sampling for multi-objective drug discovery. *arXiv preprint*, arXiv:2103.10432, 2021.

[56] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[57] Yulei Yang and Dongsheng Li. NENN: incorporate node and edge features in graph neural networks. In *Proceedings of The 12th Asian Conference on Machine Learning, ACML 2020, 18-20 November 2020, Bangkok, Thailand*, volume 129 of *Proceedings of Machine Learning Research*, pages 593–608. PMLR, 2020.

[58] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 4805–4815, 2018.

[59] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3391–3401, 2017.

[60] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.

[61] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint*, arXiv:1812.08434, 2018.

Appendix

Organization The appendix is organized as follows. In section A, we first describe the structural details of the proposed *Edge HyperGraph Neural Network* (EHGNN) framework using the *Dual Hypergraph Transformation* (DHT) in comparison to those of the existing edge-aware graph neural networks. Also, we describe the detailed components of the proposed edge pooling methods: *HyperCluster* and *HyperDrop* in section B. We provide the experimental setups in Section C, which include the detailed descriptions of the models and datasets, as well as the experimental details for each task. Then, we provide additional experimental results on graph reconstruction and generation tasks with visualization of examples in Section D. Finally, We discuss the limitations and potential societal impacts of our work in Section E.

A Edge Representation Learning

In this section, we first describe the detailed formulations of existing edge-aware graph neural networks (GNNs), and compare them with our methods. Then, we introduce the time complexity of making connectivity patterns for edges using existing edge-aware GNNs and our *Edge HyperGraph Neural Network* (EHGNN). Finally, we discuss the sparse implementation of the proposed EHGNN along with our *Dual Hypergraph Transformation* (DHT) at this end of the section.

A.1 Discussion on edge-aware graph neural networks

Here, we first formalize the existing edge-aware GNNs that we used as baselines [23, 17, 45, 18]. We begin by introducing the basic components of GNNs: $\boldsymbol{X}^{(l)}$ denotes the node features at l-th layer, $\boldsymbol{W}$ denotes the learnable weight matrix, $\boldsymbol{E}$ denotes the edge features, and $\mathcal{N}(v)$ denotes the neighboring node set of node v in the given graph.

EGCN The node-wise formulation of edge-aware GCN [23] is defined as follows:

$$\boldsymbol{X}_v^{(l+1)} = \boldsymbol{W} \sum_{u \in \mathcal{N}(v) \cup \{v\}} n_{u,v} \cdot (\boldsymbol{X}_u^{(l)} + \boldsymbol{E}_{u,v}) \tag{9}$$

where $n_{u,v}$ is the normalizing coefficient for two adjacent nodes u and v, and edge feature $\boldsymbol{E}$ is obtained from the categorical edge features without learning.

MPNN The node-wise formulation of MPNN [17] using edge conditioned convolution [47] is defined as follows:

$$\boldsymbol{X}_v^{(l+1)} = \boldsymbol{W}\boldsymbol{X}_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{X}_u^{(l)} \cdot \text{MLP}(\boldsymbol{E}_{u,v}) \tag{10}$$

where MLP is a linear layer for learning the edge representations to augment the node representations.

R-GCN The node-wise formulation of R-GCN [45] is defined as follows:

$$\boldsymbol{X}_v^{(l+1)} = \boldsymbol{W}\boldsymbol{X}_v^{(l)} + \sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_r(v)} \frac{1}{|\mathcal{N}_r(v)|} \boldsymbol{W}_r \boldsymbol{X}_u^{(l)} \tag{11}$$

where $\mathcal{R}$ is a set of categorical edge types, and $\mathcal{N}_r(v)$ denotes the neighboring node set of the node v, having the associated edge type $r \in \mathcal{R}$.

EGNN The node-wise formulation of convolution-based edge GNN [18] is defined as follows:

$$\boldsymbol{X}_v^{(l+1)} = \boldsymbol{W} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \boldsymbol{E}_{u,v}^{(l)} \boldsymbol{X}_u^{(l)} \tag{12}$$

where the edge features at l-th layer $\boldsymbol{E}^{(l)}$ are obtained by edge-level layers which are differently designed from node-level layers. The features are used as the attention coefficients for nodes to enhance the node-level representations.

It is worthwhile to note that all baselines only implicitly capture edge information in the learned node representations rather than directly using it for downstream graph tasks, while our EHGNN framework explicitly learns and utilizes the learned edge representations.

# of edges (10^3)	2	4	8	16	32	64
Line graph	32.78	65.93	131.36	260.92	527.44	1071.31
DHT (ours)	**0.13**	**0.18**	**0.26**	**0.45**	**0.81**	**1.37**

Table 5: **Transformation time(s) results** of the line graph and our dual hypergraph. The results are the average of over 100 runs.

Graph	Message-passing	
	Nodes (GCN)	Edges (Ours)
Erdos-Renyi	0.0031	0.0034
Barabasi-Albert	0.0029	0.0032

Table 6: **Message-passing time(s)** on the original graph and our dual hypergraph.

A.2 Sparse implementation of the dual hypergraph transformation

Since most graphs have relatively few connections per node, the number of non-zero elements in the adjacency matrix, which defines the connection among nodes, is smaller than the number of zero elements. Thus, using the adjacency matrix for message passing is highly inefficient in terms of memory usage. To handle this issue, the most dominant approach is to use an edge list, which is a sparse representation of the adjacency matrix (or the incidence matrix) of the graph. Specifically, each column of the edge list $\boldsymbol{L} \in \mathbb{R}^{2\times m}$ denotes an edge e, which has two incident nodes (v_{start}, v_{end}), where v_{start} denotes the start node and v_{end} denotes the end node of the edge e.

Similarly, the incidence matrix of a hypergraph can be represented as a sparse form using a hyperedge list $\boldsymbol{L}^* \in \mathbb{R}^{2\times D}$, where D is the sum of degrees of all nodes in the hypergraph. Each column of L^* indicates a hyperedge e^* with a (node, hyperedge) pair (v^*, e^*), where v^* is the node incident to the hyperedge e^*. If the hyperedge is incident to three nodes, then it will appear on three columns of L^* paired with each incident node. Compared to this general hypergraph, the dual hypergraph obtained by DHT is 2-regular, which means each node in the hypergraph has a degree of two, since each edge in the original graph is incident to exactly two nodes. Thanks to this property, the hyperedge list of the dual hypergraph has the dimensionality of $2 \times 2m$ (i.e., $\boldsymbol{L}^* \in \mathbb{R}^{2\times 2m}$).

Then, the concrete implementation of DHT with the sparse edge list of the original graph and the sparse hyperedge list of its dual hypergraph is formalized as follows:

$$\boldsymbol{DHT} \,:\, G = \big(\boldsymbol{X}, \boldsymbol{L}, \boldsymbol{E}\big) \;\mapsto\; G^* = \big(\boldsymbol{E}, \boldsymbol{L}^*, \boldsymbol{X}\big), \tag{13}$$

where the hyperedge list $\boldsymbol{L}^*$ is obtained by reshaping the edge list $\boldsymbol{L}$ as follows:

$$\boldsymbol{L}^*_{1,2i-1} = \boldsymbol{L}^*_{1,2i} = i, \tag{14}$$

$$\boldsymbol{L}^*_{2,2i-1} = \boldsymbol{L}_{1,i}\;, \quad \boldsymbol{L}^*_{2,2i} = \boldsymbol{L}_{2,i}, \tag{15}$$

for all $1 \leq i \leq m$.

A.3 Complexity analysis

In this subsection, we provide the detailed complexity analysis of transformation and message-passing operations of existing edge-aware GNNs [28, 18, 57] and our DHT. We first introduce the transformation complexity, and then describe the message-passing complexity.

Transformation complexity To define the adjacency of edges to perform message-passing between edges, previous works either define the edge neighborhood structure [57], or use the line graph transformation [28]. Constructing edge neighborhood takes $\mathcal{O}(m^2)$ for transforming the node adjacency to the edge adjacency, as, for verifying two edges are adjacent, we need to first sample one edge among m edges, and then find the other edge that shares the same node among the remaining $m-1$ edges. In a similar manner, the complexity of line graph transformation is quadratic to the number of edges Monti et al. [37], as, for each pair of edges, we need to verify whether they share the same node. However, with our sparse implementation of DHT explained in A.2, we can obtain the hyperedge list – a sparse data structure of the hypergraph – by simply reshaping the given edge list of the original graph, which takes at most $\mathcal{O}(m)$.

We further experimentally verify the transformation complexity of the line graph transformation [28] and the proposed DHT, on Erdos-Renyi graph [10] with 1000 nodes and the number of edges increasing from 2×10^3 to 64×10^3. As shown in Table 5, our DHT is highly efficient compared to the line graph transformation, especially for large and dense graphs, as the line graph transformation is quadratic to the number of edges, whereas ours only requires simple tensor-reshape operations.

Message-passing complexity Note that the complexity of message-passing on the graphs depends only on the number of edges, thus it is enough to focus on the number of edges. When we transform the original graph into the line graph following Jiang et al. [28], the constructed line graph has $\mathcal{O}(m \cdot d_{max})$ edges, therefore the complexity of message-passing is $\mathcal{O}(m \cdot d_{max})$. For instance, when the input graph is a star graph having one hub node and n other nodes (i.e., the number of edges is n), the line graph of the star graph has n^2 number of edges, thus the message-passing cost is $\mathcal{O}(n^2)$, as shown in Table 1 of the main paper. However, with our DHT implemented over the sparse hyperedge list, we only have $2m$ number of node-hyperedge pairs as explained in Section A.2, thus we can perform the message-passing between edges (nodes of the dual hypergraph) with complexity $\mathcal{O}(m)$. This complexity is equal to that of the message-passing between nodes of the original graph. In other words, the analytical complexity of message-passing between edges in equation 4 of the main paper is equivalent to the complexity of message-passing between nodes in equation 1 of the main paper.

We experimentally validate the message-passing complexity on the original graph (message-passing between nodes) and the dual hypergraph (message-passing between edges) in Table 6. We evaluate the message-passing time on both the Erdos-Renyi graph [10] and the scale-free (Barabasi-Albert) network [3], with 3000 nodes 11984 edges following the densification law (i.e. $m \propto n^{1.18}$ [34]) of the internet graph. Table 6 shows that message-passing time on the dual hypergraph is almost equal to the message-passing time on the original graph, which coincides with the previous analysis.

B Details for Edge Pooling Schemes

In this section, we describe the proposed two novel edge pooling schemes: *HyperCluster* that coarsens similar edges for global edge representations, and *HyperDrop* that drops unnecessary edges for hierarchical graph representations.

B.1 HyperCluster

Our cluster-based edge pooling model, HyperCluster, consists of edge-level message passing layers (i.e., EHGNN layers) and HyperCluster layers, which we describe below in detail. Before clustering edges, we first update the edge features using multiple EHGNN layers as follows:

$$\boldsymbol{E}^{(l+1)} = \text{EHGNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}^{(l)}), \tag{16}$$

where $\boldsymbol{E}^{(l)}$ denotes the updated edge features at the l-th layer from the initial edge features $\boldsymbol{E}^{(0)} = \boldsymbol{E}$, and we finally obtain $\boldsymbol{E'} = \boldsymbol{E}^{(L)}$ after L number of EHGNN layers. Then, to obtain the global edge representation of the entire graph, we cluster the nodes of its dual hypergraph using the node clustering method. While we can use any off-the-shelf node clustering methods [58, 5, 2], in this paper, we use the state-of-the-art pooling method, namely GMPool [2]. To apply GMPool on a hypergraph, we modify the graph multi-head attention block (GMH), which is used to construct key and value matrices using GNNs for the original graph structure in the GMPool paper [2], for the hypergraph structure by replacing the adjacency matrix to the incidence matrix. We compress m nodes in the dual hypergraph into k nodes with the modified GMPool$_k$, formalized as follows:

$$\boldsymbol{E}^{pool} = \text{GMPool}_k(\boldsymbol{E'}, \boldsymbol{M}^T), \quad \boldsymbol{M}^{pool} = \boldsymbol{M}\boldsymbol{C}, \tag{17}$$

where $\boldsymbol{C}$ is the cluster assignment matrix generated by GMPool. The overall architecture can be either global or hierarchical, depending on the downstream task.

B.2 HyperDrop

Our drop-based edge pooling model, HyperDrop, consists of EHGNN layers and HyperDrop layers, which we describe below in detail. Before dropping unnecessary edges, we first update the edge features using the proposed EHGNN layer as follows:

$$\boldsymbol{E'} = \text{EHGNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}). \tag{18}$$

Then, we drop the nodes of the dual hypergraph based on a learnable score function. While we can use any off-the-shelf node drop methods [15, 33] with their score functions, in this paper, we use the self-attention score based node drop method proposed in Lee et al. [33] as follows:

$$\boldsymbol{Z} = \tanh(\text{GNN}(\boldsymbol{E'}, \boldsymbol{M}^T, \boldsymbol{X})) \tag{19}$$

Based on the output score vector $\boldsymbol{Z} \in \mathbb{R}^m$ for every m nodes on the dual hypergraph, we select the top-ranked k nodes to obtain the pooled edge features and the incidence matrix as follows:

$$\boldsymbol{E}^{pool} = \boldsymbol{E'}_{idx}\,, \quad \boldsymbol{M}^{pool} = ((\boldsymbol{M}^T)_{idx})^T \;;\; idx = \text{top}_k\left(\boldsymbol{Z}\right). \tag{20}$$

Thus, we obtain the edge-pooled graph $G^{pool} = (\boldsymbol{X}, \boldsymbol{M}^{pool}, \boldsymbol{E}^{pool})$ without loss of node information of the original graph. Furthermore, we use the self-attention score vector $\boldsymbol{Z}$ as the edge weight for the node-level message passing layer, to reflect the relative importance of the neighboring information. This can be formulated as follows:

$$\boldsymbol{X'} = \text{GNN}\left(\boldsymbol{X}, \boldsymbol{M}^{pool}, \boldsymbol{Z}_{idx}\right), \tag{21}$$

where we can use simple GCN [32] or edge-aware GNNs for the GNN function.

C Experimental Setup

In this section, we introduce baselines and proposed models that we used for verifying the effectiveness of our approaches, in two different paragraphs: one for message passing methods and another for graph pooling methods, and then provide the information of the computing resources. After that, we describe the experimental details about four different tasks on which we validate our methods.

Baselines and our model for graph neural networks Here, we describe a set encoding model that ignores connectivity between nodes, naive graph neural networks that only consider node features without edge information, edge-aware graph neural networks that use edge features as auxiliary information for updating node features, and our model that explicitly represents edges as follows:

1. **DeepSet.** This method [59] is the set encoding baseline that first represents each node with a linear function, and then aggregates all node representations with sum pooling, which does not consider connectivity patterns between nodes.
2. **GCN.** This method [32] is the naive graph neural network baseline that aggregates neighboring nodes' information using the mean operation, which does not consider edge information. Also, we obtain the entire graph representation using the mean pooling of all nodes.
3. **GIN.** This method [56] is the naive graph neural network baseline that aggregates neighboring node's information using the sum operation, which does not consider edge information. Also, we obtain the entire graph representation using the sum pooling of all nodes.
4. **EGCN.** This method [22] is the edge-aware graph neural network baseline that uses edges as auxiliary information only to augment node-level representations, by adding the edge features between a node and its neighborhood to the node features (see Section A.1 for detailed formulation).
5. **MPNN.** This method [17] is the edge-aware graph neural network baseline that uses edges as auxiliary information only to augment the node-level representations, by multiplying the edge features between a node and its neighborhood to the node feature (see Section A.1 for details).
6. **R-GCN.** This method [45] is the edge-aware graph neural network baseline that uses discrete edge features for considering relation types between nodes, by multiplying the categorical weights of edges to the node features (see Section A.1 for detailed formulation).
7. **EGNN.** This method [18] is the edge-aware graph neural network baseline that first obtains explicit edge representations using differently designed edge-level layer, and then uses them to augment node-level representations, by multiplying the edge representations to the node representations (see Section A.1 for detailed formulation).
8. **EHGNN.** This is our edge representation learning framework that first transforms the given original graph into its dual hypergraph with *Dual Hypergraph Transformation*, and then obtain the explicit edge representations with existing off-the-shelf message-passing schemes for nodes, which is further directly used for graph-level representation learning.

Baselines and our model for graph pooling Here, we explain the global node pooling baselines, as well as the hierarchical node pooling baselines. Then, we describe the proposed two novel edge pooling schemes: cluster-based and drop-based methods, for graph-level representation learning.

1. **DiffPool.** This method [58] is the hierarchical node pooling baseline that coarsens nodes with a clustering-based approach, where it generates a cluster-assignment matrix for nodes using a GNN.

2. **SAGPool.** This method [33] is the hierarchical node pooling baseline that drops unnecessary nodes with a drop-based approach, where it generates scores for nodes with a GNN.
3. **TopKPool.** This method [15] is the hierarchical node pooling baseline that drops unnecessary nodes with a drop-based approach, where it generates scores for nodes with MLPs.
4. **MinCutPool.** This method [5] is the hierarchical node pooling baseline that coarsens nodes with a clustering-based approach, where it generates a cluster-assignment matrix for nodes using MLPs.
5. **ASAP.** This method [42] is the hierarchical node pooling baseline that first clusters similar nodes, then drop unnecessary clusters to coarsen an entire graph.
6. **EdgePool.** This method [8] is the hierarchical node pooling baseline that computes the edge score between nodes, then contracts two adjacent nodes with the high edge score into a single node.
7. **HaarPool.** This method [52] is the hierarchical node pooling baseline that coarsens nodes with the Haar transformation, which is based on the Haar basis in the Haar wavelet domain [51].
8. **SortPool.** This method [60] is the global node pooling baseline that first sorts the obtained node representations at the end of graph convolution layers, then predicts an entire graph representation with sorted node features.
9. **GMPool.** This method [2] is the global node pooling baseline that uses self-attention based operations to compress multiple nodes into a few clusters with learnable cluster assignment vectors to obtain an entire graph representation.
10. **GMT.** This method [2] is the global node pooling baseline that stacks self-attention based layers not only to compress many nodes into a few clusters with learnable cluster assignment vectors, but also to consider the inter-node (or cluster) relationships to obtain an entire graph representation.
11. **HyperCluster.** This is our global edge representation learning scheme that coarsens similar edges into a single edge to obtain a holistic edge-level representation, where we can generate the cluster assignment matrix for edges using existing clustering-based methods, such as GMPool [2] (see Section B.1 for more details).
12. **HyperDrop.** This is our hierarchical edge representation learning scheme that drops unnecessary edges based on a learnable score function, such as MLPs or GNNs, thereby adjusting the graph topology for more effective message passing. Notably, this scheme does not result in the removal of any nodes. (see Section B.2 for more details).

Computing resources For all experiments, we use PyTorch [40] and PyTorch geometric [14], and train each model on a single Titan XP, GeForce GTX Titan X, or GeForce RTX 2080 Ti GPU. A single experiment of each task takes less than 1 day, and for the classification tasks such as node or graph classification, the single runtime on most datasets of a relatively small size is less than 1 hour.

C.1 Graph reconstruction

Common implementation details Given a set of graphs $\{G = (\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E})\}$, the goal of graph reconstruction is to reconstruct both node and edge features from the compressed representations, by training two separate autoencoders where one is trained for reconstructing node features and the other is trained for reconstructing edge features. Formally, we define the node and edge encoders as ENC_{node} and ENC_{edge}, respectively, and the node and edge decoders as DEC_{node} and DEC_{edge}, respectively. Then, following the standard architecture setting of graph reconstruction tasks of existing works [5, 2], the node-level autoencoder which is a pair of the node encoder and node decoder, ENC_{node} and DEC_{node}, is defined as follows:

$$\text{ENC}_{node}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}) = \text{GMPool}(\text{GNN}(\text{GNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}))) = \boldsymbol{X}^{pool}, \tag{22}$$

$$\text{DEC}_{node}(\boldsymbol{X}^{pool}, \boldsymbol{M}, \boldsymbol{E}) = \text{GNN}(\text{GNN}(\text{GNN}(\text{GMPool}^{-1}(\boldsymbol{X}^{pool}, \boldsymbol{M}, \boldsymbol{E})))) = \boldsymbol{X}^{rec}, \tag{23}$$

where we use the GMPool [2] for reconstructing node features, as it shows outstanding performance on node-level reconstruction tasks. GMPool denotes the pooling operation, and GMPool^{-1} denotes the unpooling operation following the setting of the original paper [2]. Also, $\boldsymbol{X}^{rec} \in \mathbb{R}^{n \times d}$ is the reconstructed node features from the pooled node representations $\boldsymbol{X}^{pool} \in \mathbb{R}^{k \times d}$, where k is the number of pooled nodes and n is the number of all nodes. We omit the inputs of the GNN, which are the incidence matrix $\boldsymbol{M}$ and the edge feature matrix $\boldsymbol{E}$, for simplicity.

However, to reconstruct the entire graph which have both node and edge features, we further need to define a separate edge-level autoencoder. Thus, similarly to the node-level autoencoder, we define the edge-level reconstruction module as a pair of the edge encoder and edge decoder, ENC_{edge} and DEC_{edge}, formalized as follows:

$$\text{ENC}_{edge}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}) = \text{Pool}(\text{GNN}(\text{GNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}))) = \boldsymbol{E}^{pool}, \tag{24}$$

$$\text{DEC}_{edge}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}^{pool}) = \text{GNN}(\text{GNN}(\text{GNN}(\text{Pool}^{-1}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}^{pool})))) = \boldsymbol{E}^{rec}, \tag{25}$$

where, for our models, we use the EHGNN with the GCN [32] for GNN operations, and HyperCluster for pooling and unpooling operations which is described in Section B.1 in detail. Meanwhile, for the baselines, we use the existing edge-aware GNNs [23, 17, 45, 18] for GNN operations, and GMPool [2] for pooling and unpooling operations, where we obtain the final edge representation by averaging the two representations of incident nodes for the edge. This is because the baselines only use edge features as auxiliary information for updating node features. $\boldsymbol{E}^{rec} \in \mathbb{R}^{m \times d'}$ is the reconstructed edge features from the pooled edge representations $\boldsymbol{E}^{pool} \in \mathbb{R}^{k' \times d'}$, where k' is the number of pooled edges and m is the number of all edges. Similar to the formulation of the node-level autoencoder, we omit the inputs of the GNN for simplicity.

Our reconstruction objective is to minimize the discrepancy between the original graph $G = (\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E})$ and the reconstructed graph $G^{rec} = (\boldsymbol{X}^{rec}, \boldsymbol{M}, \boldsymbol{E}^{rec})$, with a loss function such as mean squared error or cross-entropy loss for node and edge features. For the edge reconstruction task, we only use the edge autoencoder without using the node autoencoder. For all reconstruction experiments, the learning rate of the node autoencoder is set to 5×10^{-3}, and the learning rate of the edge autoencoder is set to 1×10^{-3}. We optimize the full network using an Adam optimizer [31].

Implementation details on synthetic graphs For the edge reconstruction of a synthetic graph, we use the standard two-moon graph generated by the PyGSP library [7], with node features given by their coordinates and edge features given by RGB colors of which values range from 0 to 1. Then, the goal of the edge reconstruction task is to restore all edge colors from the compressed edge representations after edge pooling. To minimize the discrepancy between original and reconstructed edge features, we use the mean squared error loss as the learning objective. Also, we use the early stopping criterion, where we stop the training if there is no further improvement on the training loss during 1,000 epochs, and the maximum number of epochs is set to 5,000. We set the pooling ratio of all models as 1% with the hidden dimension of size 16.

Implementation details on molecular graphs Following the experimental setting of the existing work [9, 2], we use the subset of the full ZINC dataset [27], which consists of 12K molecular graphs, where node features are atom types and edge features are bond types. The number of atom types is 28, and the number of bond types is 5. We follow the dataset splitting of training, validation, and test sets from Dwivedi et al. [9]. Then, the goal of the molecular graph reconstruction task is to restore both atom types and bond types of all nodes and edges from their compressed representations after pooling. To train the model, we use the cross-entropy loss for molecular graph reconstruction, since the initial features given for nodes and edges are discrete. We also use the early stopping criterion, where we stop the training if there is no further improvement on the validation loss during 200 epochs. For hyperparameters, the maximum number of epochs is set to 500, hidden dimension size is set to 32, and batch size is set to 128. We run five experiments with different random seeds, and report the average performance with its standard deviation. Following the evaluation setup of Baek et al. [2], we use the following three metrics: *accuracy* measures the classification accuracy of all nodes and edges, *validity* counts the number of reconstructed molecules which are chemically valid, and *exact match* counts the number of reconstructed molecules which are identical to the original molecules.

Implementation details on graph compression We quantitatively compare the relative memory size of the compressed graph after pooling nodes and edges against the size of the original graph, which we use the Erdos-Renyi random graph model [10]. We compare our proposed method EHGNN with HyperCluster, with the node pooling baseline, GMT. The number of nodes is fixed to 10^3, while the number of edges is selected from one of 10^3, 5×10^3, and 10^4. To obtain the features of nodes and edges, we first randomly assign one of three values to each node (i.e., one among $\{0, 1, 2\}$), and then generate edge features using the values of two adjacent nodes for each edge. For example, if two nodes have the same 0 value for the incident edge, then we assign the zero value to the edge feature. Since the total number of pairs of node values is six for the undirected graph, the number of edge features is six. The node pooling ratio is equally fixed to 15% for both GMT and our model, and we report the relative size of the entire graph with the edge reconstruction accuracy higher than 95% or 75%, where the edge pooling ratio is decided according to its accuracy.

C.2 Graph generation

Implementation details on MolGAN architectures We use the QM9 dataset [41] that contains 133,885 organic compounds, where each molecular graph consists of carbon (C), oxygen (O), nitrogen (N), and fluorine (F) with up to nine non-hydrogen atoms. To evaluate the generated molecular graphs, we use the normalized Synthetic Accessibility (SA) and Druglikeness (QED) scores following the evaluation setup of the original paper [6]. Also, we use the categorical re-parameterization trick with the Gumbel-softmax function during the discretization process of molecule generation, to train the model in an end-to-end fashion, which adapts the learning scheme of the original paper [6].

In the original MolGAN [6], R-GCNs [45] are used to encode feature representations of nodes for the discriminator and reward networks. Learning rates of the generator, the discriminator, and the reward network are equally set to 1×10^{-3}, and hidden sizes of the two-layer R-GCNs are 128 and 64. For the MolGAN with GMPool (MolGAN + GMPool) setting, the GMPool, which is the global node pooling baseline, is additionally used to obtain the compact node-level representations. The tanh activation function is used for GMPool. For the MolGAN with the proposed EHGNN (MolGAN + EHGNN) setting, we use two EHGNN layers to encode the feature representations of the edges, wherein we use the GCN as the edge-level message-passing function. The hidden sizes are set to 32 and 16. After obtaining the edge-level representations, we use mean pooling to obtain the global edge representation, which is forwarded to the discriminator and reward networks. We further combine the GMPool with the MolGAN + EHGNN combination (MolGAN + GMPool + EHGNN) to additionally enhance the global graph representation with both node and edge representations. The learning rate of the EHGNN parameters in the discriminator and reward networks is set to 1×10^{-2}. Also, all the models use Adam optimizer [31] for training. Regarding other settings, we strictly follow the original MolGAN paper [6], and use the available code[3].

Implementation details on MARS architectures For the experiments using the MARS architecture, we use the ZINC15 [48, 24] dataset, which contains 2 million molecules, and we use the available data[4] from Hu et al. [24]. Further, we provide additional experimental results on the ChEMBL [16] dataset, which consists of 1,488,640 molecules, in Section D.2. As the fragments of molecular graphs are the basic building blocks for molecular graph generation in the MARS [55], we build the fragment vocabularies following the same procedure of the original MARS paper: fragments are built by breaking a single bond of molecules from the given dataset, limiting the size of fragments to 10 atoms (see the original paper [55] for more details on the generation process of fragment vocabularies). We report the statistics of generated fragments from each dataset in Table 7.

Table 7: Statistics of fragment vocabularies of ZINC15 and ChEMBL datasets on MARS experiments.

	ZINC15	**ChEMBL**
# of node types	9	9
Avg # of nodes	7.68	7.35
# of edge types	4	4
Avg # of edges	7.54	7.08

The MARS model sequentially generates molecules by taking one of the addition or deletion actions at each step, especially where this model uses the explicit edge representation on the deletion actions. For a set of given graphs $\{G = (\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E})\}$, the original MARS model obtains the edge representation for the deletion actions as follows:

$$\begin{aligned} \boldsymbol{X'} &= \text{MPNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}) \\ \boldsymbol{E}'_e &= \text{Concat}(\boldsymbol{X}'_u, \boldsymbol{X}'_v, \text{MLP}(\boldsymbol{E}_e)) \end{aligned} \tag{26}$$

where MPNN is the edge-aware graph neural network described in the subsection C, an edge e is incident to two nodes u and v, and $\boldsymbol{E}'_e$ is the output edge representation of the edge e. Compared to this baseline that implicitly captures the edge representation on the learned node representation $\boldsymbol{X'}$ with the concatenated edge representation through the naive MLP layer, for our model, we replace the MLP layer with the proposed EHGNN to explicitly learn the edge representation via edge-level message passing. For a fair comparison in terms of the number of parameters, we use the same number of layers and embedding size for both MLP and EHGNN.

[3] https://github.com/yongqyu/MolGAN-pytorch

[4] http://snap.stanford.edu/gnn-pretrain/data/

Following the experimental setup of the original MARS paper [55], we train the models to maximize the sum of multiple scores: QED, SA, and target protein inhibition scores against GSK3β and JNK3, respectively. For evaluation metrics, we measure the percentage of the generated molecules having scores above a certain threshold for each property: QED ≥ 0.67, SA ≥ 0.67, and the inhibition scores against GSK3$\beta \geq 0.6$ and JNK3 ≥ 0.6. The success rate can measure the overall multi-objective score by calculating the percentage of the generated molecules satisfying all four objectives. We also report the suggested easier threshold from the original MARS paper [55]: QED ≥ 0.6, SA ≥ 0.67, and the inhibition scores against GSK3$\beta \geq 0.5$ and JNK3 ≥ 0.5, in Section D.2, where we see the same tendency for the results of baseline and our model. For the experiment on ZINC15, we set the learning rate of EHGNN parameters to 5×10^{-3} with a cosine scheduler for learning rate warmup. For the experiment on ChEMBL, we set the learning rate of EHGNN parameters to 3×10^{-4}. The learning rate of other parameters in MARS is set to 3×10^{-4}, following the original paper [55]. We use the available code[6] from the original MARS paper.

C.3 Graph classification

Datasets We validate our models on ten different benchmark datasets including six from the TU datasets [38] and four from the OGB datasets [22]. For a fair comparison of baselines and our model, following the standard experimental setting of Errica et al. [12], we use the one-hot encoding of atom types as initial node features in TU bio-chemical datasets (D&D, PROTEINS, MUTAG) and one-hot encoding of node degrees as initial node features in TU social datasets (IMDB-B, IMDB-M, COLLAB), if initial node features are not given in advance. Furthermore, if the initial edge features are not given in advance, we set them to one uniformly. For the dataset splitting of the TU datasets, we follow the standard training/test splits from Niepert et al. [39], Zhang et al. [60], Baek et al. [2], and further divide the training set into training and validation sets by using the 10 percent of the training data as validation data, as suggested by the fair comparison setup of Errica et al. [12]. For the OGB datasets (HIV, Tox21, ToxCast), following the original dataset paper [23], we use the additional atom and bond features for each graph, and follow the performance evaluation and data split setting of Hu et al. [23]. The statistics of each dataset are provided in Table 3 of the main paper.

Implementation details We follow the standard experiment setting from Baek et al. [2] with the same base architectures and hyperparameters for all models on all datasets[5]. Notably, we stack three number of GCN layers as node-level message passing for all pooling models, including ours. For our model, we use the GCN for the EHGNN layer, where we equally stack three number of EHGNN layers to obtain the explicit edge representations, in parallel with node-level layers. Also, from the explicitly learned edge representations, we drop edges with their scores at each edge-level layer, which is described in section B.2 in detail. For the model HyperDrop + GMT, we apply the global node pooling layer GMPool [2] after the HyperDrop layers to obtain the global representation. For the hyperparameters of our HyperDrop, we set the hidden dimension of edges as 128 except the COLLAB dataset, on which we set the hidden dimension as 16, since the COLLAB dataset has a large number of edges compared to other datasets. Also, we randomly search for the edge drop ratio by increasing the drop ratios from 5% to 75% with 5% increments. We report the average performances and standard deviations of 10 runs with different random seeds on test datasets.

C.4 Node classification

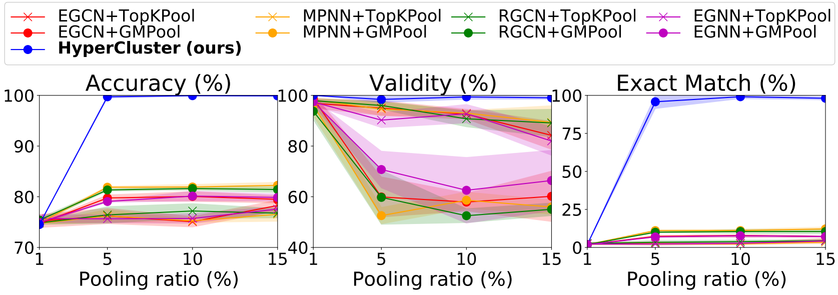
To demonstrate HyperDrop's effectiveness in alleviating the over-smoothing problem in deep GNNs, we validate it on the semi-supervised node classification tasks.

Datasets We experiment on two benchmark datasets [46], namely Cora and Citeseer, which is the citation network where nodes are documents and edges are citation links between documents. The goal of the node classification task is to predict the class of the documents (nodes). The Cora dataset consists of 2,708 nodes and 5,429 edges with 7 classes. Also, the Citesser dataset consists of 3,327 nodes and 4,732 edges with 6 classes. Node features for each dataset consist of bag-of-words for each document. As the initial edge features are not given, we set them by concatenating the features of two endpoints of the edge. We use the classification accuracy as an evaluation metric.

Implementation details For a fair evaluation of the semi-supervised node classification task, we follow the standard experimental setting of existing works [32, 50, 13], from the node features to the

[5] https://github.com/JinheonBaek/GMT

Figure 11: **Additional edge reconstruction results with TopKPool** on the ZINC dataset by varying the compression ratio. Along with the results of Figure 3 in the main paper, we additionally report the average performance of the baselines using TopKPool over 5 different runs with the standard deviation.



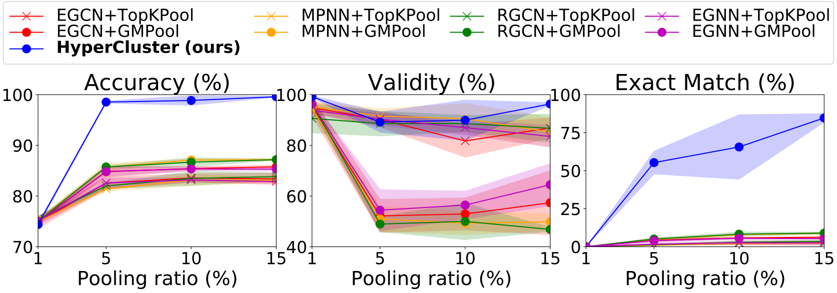


Figure 12: **Additional graph reconstruction results with TopKPool** on the ZINC dataset by varying the compression ratio. Along with the results of Figure 5 in the main paper, we additionally report the average performance of the baselines using TopKPool over 5 different runs with the standard deviation.

dataset splitting. Regarding baselines, we use the naive GCN [32], GCN with batch normalization [26], and random edge drop scheme [43]. Specifically, for the GCN with batch normalization, we use the batch normalization layer between every GCN layer to normalize the features of nodes. Also, for the random edge drop baseline, we randomly drop the partial number of edges before the first layer of GNNs, following the setting of Rong et al. [43], where we do not use the batch normalization to directly see the effect of random drop on the over-smoothing problem. For our model, we use the HyperDrop with EHGNN (see section B.2 for detailed architectures), where we drop edges when passing through every four GNN layers starting from the second layer, and we do not use the batch normalization. Finally, we use the GCN as the node-level message passing layers for all models, and also use it as the edge-level message passing layers for our HyperDrop with EHGNN.

Following the hyperparameters of the existing semi-supervised node classification work [13], for the Cora dataset, we set the dropout rate as 0.5, hidden size as 32, and learning rate as 0.01. Also, for the Citeseer dataset, we use the same setting from the Cora dataset except for the dropout rate which is set to 0.2. For the random drop and our models, we drop 20% of edges at each drop step.

D Additional Experimental Results

In this section, we provide the additional experimental results on graph reconstruction and generation tasks, with examples of reconstructed or generated molecules. Then, to further qualitatively evaluate the performances of our model, we visualize the edge pooling process of the proposed HyperDrop.

D.1 Graph reconstruction

Additional graph reconstruction results To see the effect of the pooling method on edge and graph reconstruction tasks, we additionally provide the performance of the TopKPool, a representative node drop method, with existing edge-aware GNN baselines as well as the performance of the GMPool, a node clustering method used in our main paper. For the comparison of the pooling methods, we report the performances of both TopKPool and GMPool, in Figure 11 for edge reconstruction and in Figure 12 for graph reconstruction. As shown in Figure 11 and Figure 12, the proposed EHGNN with HyperCluster largely outperforms all the baselines, which suggests that accurately

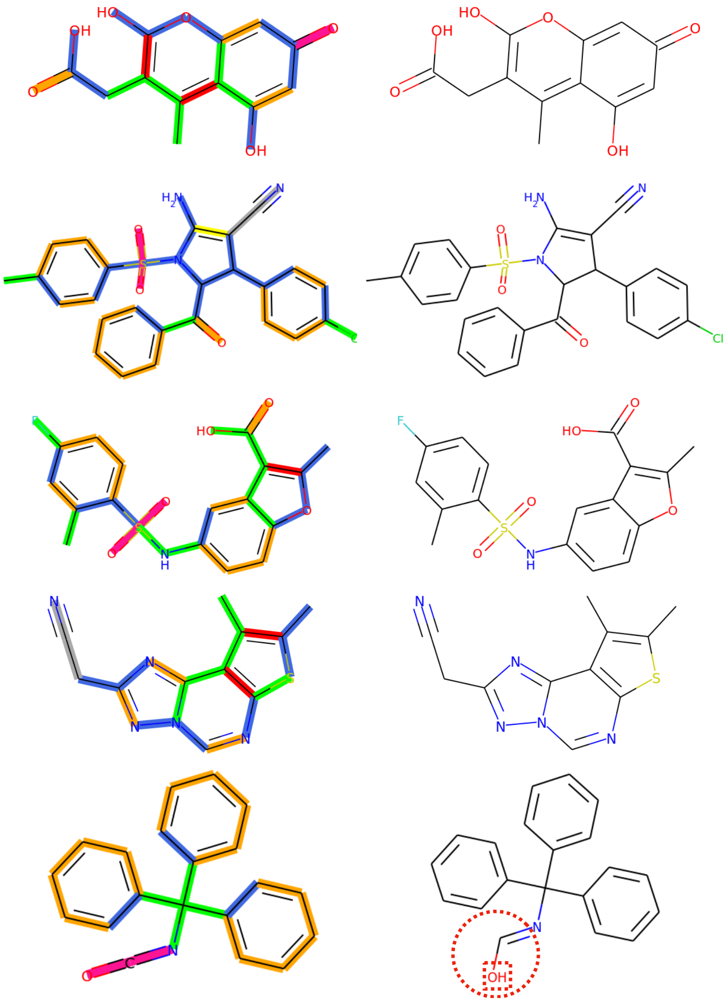
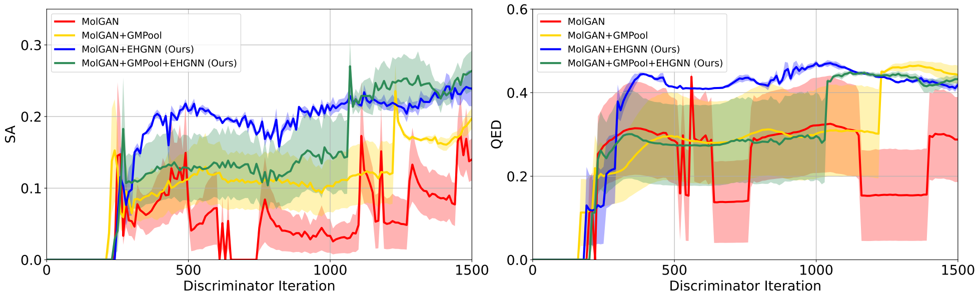


Figure 13: **Molecule reconstruction examples.** Molecules shown in the left column are the original molecules with an assigned cluster on each edge, where each cluster is represented as color. The clusters are generated by our method, HyperCluster. The molecules shown in the right column are the reconstructed molecules with our method, where red circles and squares indicate the incorrect prediction of edges and nodes, respectively.

learning the edge representations is more important than choosing which pooling methods to use, in order to obtain the global graph-level representations. Moreover, we observe that the node drop method (TopKPool) for reconstruction is inferior to the node clustering method (GMPool) in terms of accuracy and exact match, since drop methods result in the removal of nodes and edges. The performance gain in validity with the TopKPool mostly comes from its reconstruction of a graph with a single bond, which makes them valid but far different from the desired reconstructed molecules.

Additional examples of molecular graph reconstruction We provide additional examples of reconstructed molecular graphs on the ZINC dataset in Figure 13. Molecules on the left side are the original molecules with each edge color indicating the assigned cluster, obtained by our HyperCluster. Molecules on the right side are the reconstructed molecules, where red circles and squares denote the incorrect predictions of edges and nodes, respectively. As shown in Figure 13, we can see that the clusters are meaningfully assigned with respect to the underlying substructures considering both edges and nodes. For example, edges in the hexagonal ring are assigned to orange and blue colors, where their color patterns are generally determined by the number of adjacent edges with their bond type. Moreover, triple bonds connected to the nitrogen (N) are assigned to the silver-colored cluster.

Figure 14: **Graph generation results on MolGAN.** Along with the results of Figure 8 in the main paper, we additionally report the performance of the combination of MolGAN, EHGNN, and GMPool. Solid lines denote the mean, and shaded areas denote the standard deviation of 3 different runs.



Datasets	Metrics	MARS	MARS + EHGNN (Ours)
ZINC15	Success Rate	59.53 ± 2.11	**64.30** ± 1.54
	QED (≥ 0.67)	95.71 ± 0.09	**96.36** ± 0.49
	SA (≥ 0.67)	**99.99** ± 0.01	**99.99** ± 0.02
	GSK3β (≥ 0.6)	86.52 ± 1.67	**90.63** ± 2.57
	JNK3 (≥ 0.6)	71.52 ± 4.15	**73.60** ± 1.29
ChEMBL	Success Rate	56.64 ± 5.79	**58.25** ± 6.07
	QED (≥ 0.67)	91.01 ± 2.79	**91.13** ± 4.84
	SA (≥ 0.67)	99.99 ± 0.01	**100.00** ± 0.00
	GSK3β (≥ 0.6)	87.45 ± 1.73	**90.34** ± 2.65
	JNK3 (≥ 0.6)	**70.57** ± 4.75	70.01 ± 4.83

Table 8: **Graph generation results on MARS including all evaluation metrics.** The results are the mean and standard deviation of 3 runs.

Datasets	Metrics	MARS	MARS + EHGNN (Ours)
ZINC15	Success Rate	95.65 ± 0.90	**97.28** ± 1.14
	QED (≥ 0.6)	99.07 ± 0.29	**99.45** ± 0.15
	SA (≥ 0.67)	**99.99** ± 0.01	**99.99** ± 0.02
	GSK3β (≥ 0.5)	99.13 ± 0.12	**99.52** ± 0.23
	JNK3 (≥ 0.5)	97.33 ± 1.30	**98.21** ± 0.89
ChEMBL	Success Rate	**92.03** ± 3.83	91.88 ± 3.50
	QED (≥ 0.6)	**96.76** ± 1.44	96.43 ± 2.83
	SA (≥ 0.67)	99.99 ± 0.01	**100.00** ± 0.00
	GSK3β (≥ 0.5)	99.19 ± 0.31	**99.39** ± 0.23
	JNK3 (≥ 0.5)	95.83 ± 2.30	**95.85** ± 0.92

Table 9: **Graph generation results on MARS under the setting of original success thresholds.** The results are the mean and standard deviation of 3 runs.

D.2 Graph generation

MolGAN Since the EHGNN framework can be jointly used with the node-level representation learning methods, we can further combine the EHGNN framework with the node pooling method, for obtaining holistic graph-level representation from both node and edge representations. Thus, we additionally couple the MolGAN + EHGNN with the state-of-the-art node pooling method, namely GMPool. As shown in Figure 14, compared to the large performance gain obtained by our EHGNN, the performance gain obtained from using both GMPool and EHGNN is relatively small, and also the training using both architectures is unstable. This might be because, we can already obtain the effective graph-level representation only with the combination of MolGAN and EHGNN, and additionally using more layers makes the training of the MolGAN architecture difficult since this scheme also increases the number of parameters. On the other perspective, since the original MolGAN architecture is already able to utilize the node representations, albeit, by simple R-GCN, the remaining performance gain comes from the explicit edge representations via our EHGNN.

MARS Here, we provide the additional experimental results using the MARS architecture on the ChEMBL dataset, where we used the available data[6] from Xie et al. [55]. As shown in Table 8, MARS equipped with our EHGNN outperforms the baseline model, showing the same tendency as in the results on the ZINC15 dataset. Also, the original MARS and the MARS with EHGNN models successfully generate the high-quality molecules in terms of SA, and there is not much significant difference between those two models on this metric. However, the performance gain with our EHGNN against the naive MARS comes from other metrics, such as QED and GNK3β, resulting in the successful generation of molecules having all desired properties.

On the other hand, we also report the success rate with individual evaluation metrics according to thresholds used in the MARS paper [55] in Table 9. As shown in Table 9, our MARS + EHGNN model still outperforms the baseline on most of the metrics, and the performance tendency is highly similar to the result of different thresholds in Table 8. Those two results demonstrate that accurate learning of edge representation is important to generate desirable molecules.

[6] https://github.com/yutxie/mars

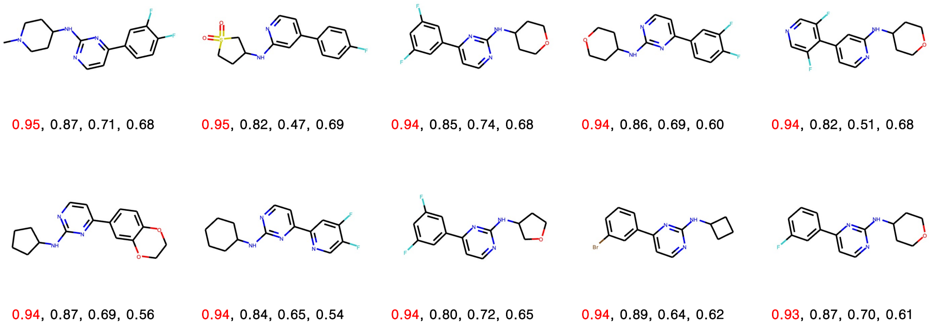


Figure 15: **10 generated molecules with the highest QED scores.** The numbers are QED, SA, GSK3β, and JNK3 scores, respectively. We highlight the QED score in red among four different scores.

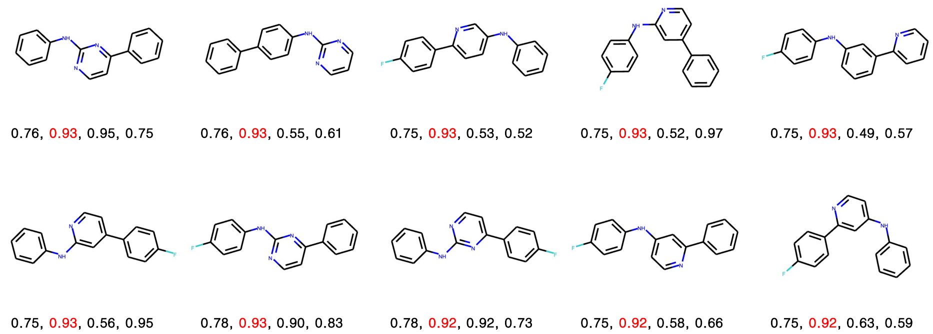


Figure 16: **10 generated molecules with the highest SA scores.** The numbers are QED, SA, GSK3β, and JNK3 scores, respectively. We highlight the SA score in red among four different scores.

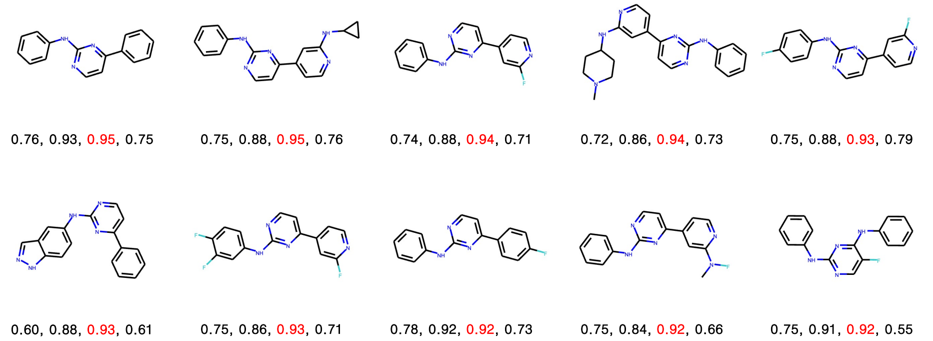


Figure 17: **10 generated molecules with the highest GSK3β scores.** The numbers are QED, SA, GSK3β, and JNK3 scores, respectively. We highlight the GSK3β score in red among four different scores.

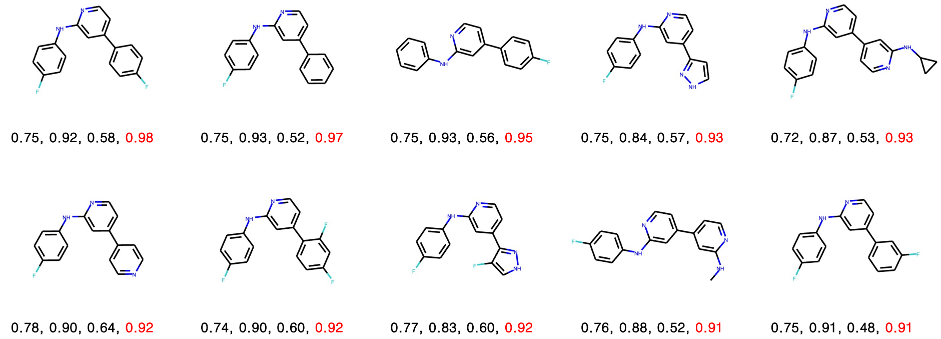


Figure 18: **10 generated molecules with the highest JNK3 scores.** The numbers are QED, SA, GSK3β, and JNK3 scores, respectively. We highlight the JNK3 score in red among four different scores.

Visualization of the generated molecular graphs We further provide the examples of generated molecules using our EHGNN on MARS in Figure 15, 16, 17, and 18. We hope that these examples are to be helpful for the chemists to get an insight into the molecules generated with our framework.

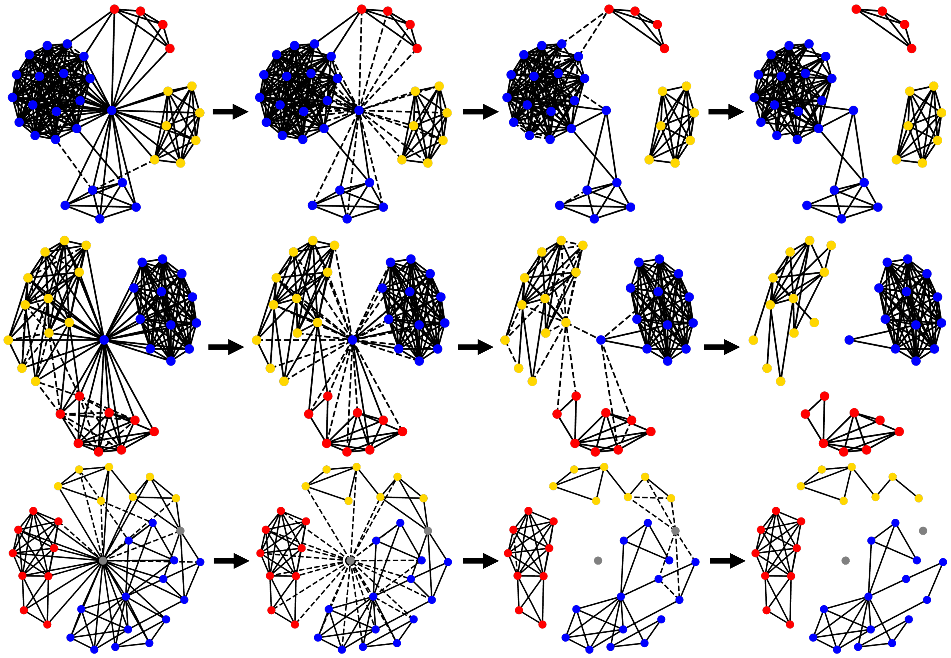


Figure 19: **Edge pooling results on the COLLAB dataset.** Each row represents the pooling process of a graph. Colors denote connected components.

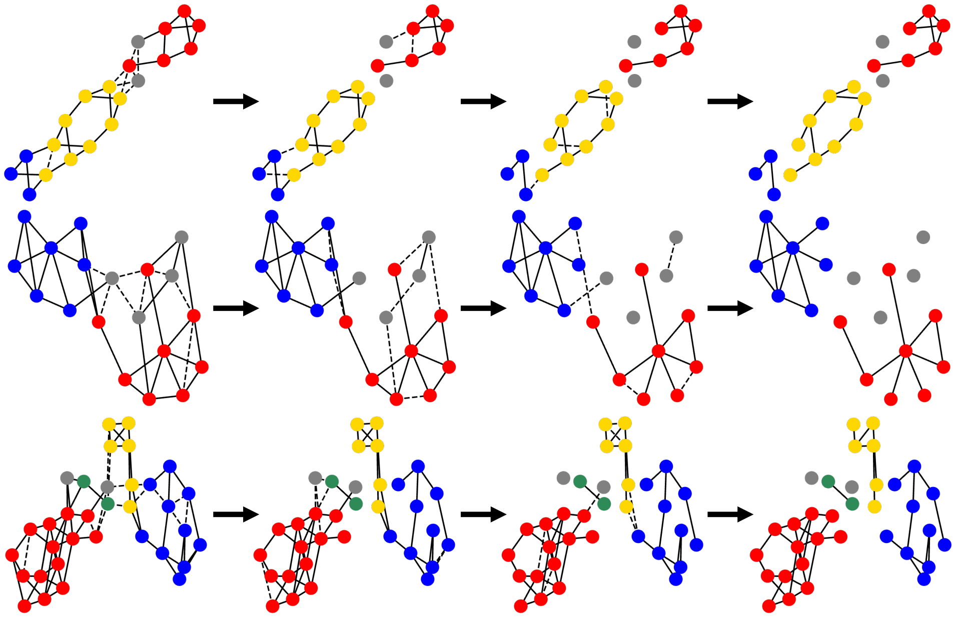


Figure 20: **Edge pooling results on the PROTEINS dataset.** Each row represents the pooling process of a graph. Colors denote connected components.

D.3 Graph classification

Additional examples of HyperDrop process We provide additional examples of HyperDrop processes on the COLLAB and PROTEINS datasets in Figure 19 and Figure 20, respectively. Colors represent the resulting connected components in the final graph after dropping edges, and we represent isolated nodes as gray. Arrows indicate the layer-wise progressive pooling processes. We can see that by dropping unnecessary edges, a large graph is divided into smaller connected components, which we assume to be effective for message passing between the relevant nodes.

E Limitations and Potential Societal Impacts

In this section, we discuss the limitations and potential societal impacts of our work.

Limitations In this work, we propose to learn edge representations with hypergraphs, using the dual hypergraph transformation that allows us to apply off-the-shelf node-level message-passing schemes designed for node representation learning to edges. While we can learn accurate edge representations using the proposed framework, we need two separate GNNs to learn node and edge representations independently. Combining these two GNNs into one, by learning node and edge representations jointly using a single GNN, may be more effective for learning graph representations, while saving the memory as well. We leave this as future work.

Potential societal impacts The system for generating target molecules is significantly important to our society, since it can be used to generate vaccines or drugs for diseases, even for the newly emerged severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). However, the conventional development of beneficial molecules requires a huge amount of time and resources with a significant number of trial-and-error processes, before actually applying the generated molecules, since we have to check potential outcomes those molecules can have.

In this paper, we show that the proposed edge representation learning framework can accurately represent the edges of the given graph, for the holistic graph-level representation learning, which has been extensively validated on graph generation and classification tasks with biochemical molecules. Therefore, this approach can meaningfully aid the development of target molecules in the following ways. First, the generation system described in Section 4.2 of the main paper is effective for generating molecules with desirable properties, since it can generate more drug-like molecules that can effectively inhibit multiple target proteins. Also, the classification system described in Section 4.3 of the main paper is beneficial for examining the toxicity of generated molecules, which is an essential step before human clinical trials or being deployed on a commercial scale. Therefore, our method allows us to reduce time and resources for generating and validating target molecules, for example in the domain of de novo drug design compared to synthesizing drugs by trial-and-error.

As described above, while our method has huge potential impacts for discovering novel molecules in our real-life, anyone can maliciously use our system, aiming to develop harmful compounds for humans, such as synthesizing toxic or addictive substances. Thus, we strongly hope that our method would not be applied for generating harmful molecules that may have negative impacts on our society.