



OPEN

# Graph representation learning via enhanced GNNs and transformers

Hongrui Mu<sup>1,3</sup>, Chengchen Zhou<sup>1,3</sup>, Qiancheng Yu<sup>1,2</sup>✉ & Qunyue Mu<sup>1</sup>

In recent years, graph transformers (GTs) have captured increasing attention within the graph domain. To address the prevalent deficiencies in local feature learning and edge information utilization inherent to GTs, we propose EHDGT, a novel graph representation learning method based on enhanced graph neural networks (GNNs) and Transformers. Initially, considering that edges encapsulate structural information, we enhance the original graph by superimposing edge-level positional encoding based on node-level random walk positional encoding, thus optimizing the utilization of this information. Subsequently, enhancements are applied to both GNNs and Transformers: for GNNs, we employ encoding strategies on subgraphs of the original graph, thereby augmenting their proficiency to process local information. For Transformers, we incorporate edges into the attention calculation and introduce a linear attention mechanism, significantly reducing the model's complexity. Ultimately, to exploit the synergies between GNNs and Transformers while maintaining a balance between local and global features, we propose a gate-based fusion mechanism for the dynamic integration of their outputs. Experimental results across multiple datasets demonstrate that EHDGT significantly outperforms traditional message-passing networks and achieves strong performance compared to several existing GTs. This paper further explores the application of the proposed model to improving the quality of the wine industry knowledge graph, experiments show that using link prediction as a downstream task of graph representation learning based on Graph Transformer achieves excellent results, significantly enhancing the completeness and semantic quality of the wine industry knowledge graph, thereby increasing its practical value in digitalization of the wine industry.

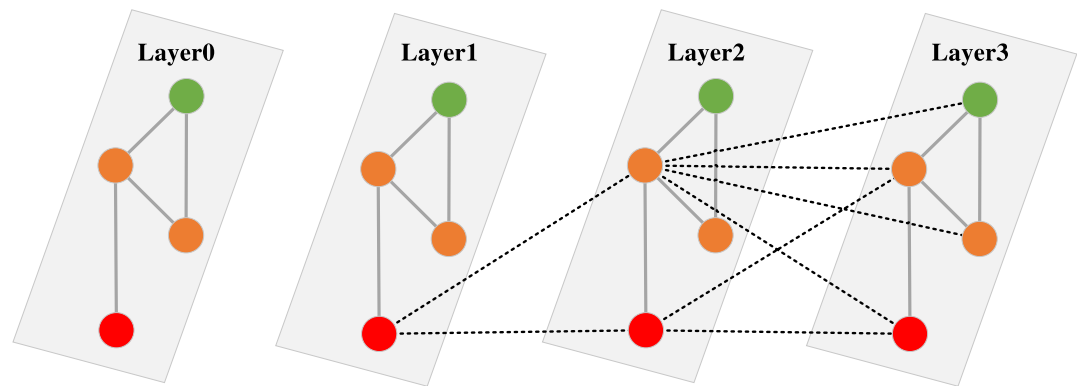
Graph-structured data consists of nodes and edges, where nodes represent objects or entities, and edges denote the relationships or connections between these objects. Such data is ubiquitous in the real world, appearing in domains like social networks, knowledge graphs, and more<sup>1</sup>. The complexity and richness of graph-structured data provide unique advantages for understanding and analyzing intricate relationships.

Graph representation learning aims to transform complex graph data into low-dimensional vector representations to enable effective analysis and mining. Unlike traditional manual feature engineering, it can automatically extract key features from graphs, significantly improving efficiency while reducing computational complexity. Moreover, the learned representations exhibit strong generality and flexibility, making them compatible with various machine learning and deep learning algorithms. This has led to their widespread application in tasks such as link prediction<sup>2</sup>, graph classification<sup>3</sup>, and graph clustering<sup>4</sup>. However, existing graph representation learning methods often struggle to effectively integrate both structural and semantic information when constructing node-level or graph-level representations. Generating more comprehensive vector representations remains one of the core focuses of graph representation learning.

GNNs<sup>5</sup> represent formidable and widely acknowledged tools for graph representation learning. As illustrated in Fig. 1, GNNs progressively aggregate neighborhood information and refine node representations through multi-layer message passing<sup>6</sup>. Despite the innovation within message-passing strategies, GNNs grapple with inherent limitations including limited expressive power<sup>7,8</sup>, propensity for over-smoothing<sup>9–11</sup>, and tendencies towards over-squashing<sup>12</sup>. Over-smoothing manifests as node representations becoming excessively similar and difficult to distinguish after multiple layers. Over-squashing reflects the challenge that messages from distant nodes encounter ineffective transmission, as excessive data is condensed into limited vector space. To mitigate these constraints, the development of architectures that surpass conventional neighborhood aggregation strategies has become imperative<sup>13</sup>.

Over the years, Transformers<sup>14</sup> have been successfully applied across three of the most significant domains in artificial intelligence: natural language processing, computer vision, and audio processing. Transformers enable efficient processing of large-scale data through parallel computation and excel at capturing global information

<sup>1</sup>School of Computer Science and Engineering, North Minzu University, Yinchuan 750021, China. <sup>2</sup>The Key Laboratory of Images and Graphics Intelligent Processing of State Ethnic Affairs Commission, North Minzu University, Yinchuan 750021, China. <sup>3</sup>Hongrui Mu and Chengchen Zhou contributed equally to this work. ✉email: 1999019@nmu.edu.cn



**Fig. 1.** Message-passing networks.

with reduced inductive bias, which has led to their widespread adoption<sup>15</sup>. Additionally, recent advancements have seen the development of numerous Transformer variants tailored for processing graph data, collectively known as Graph Transformers (GTs). In many cases, these variants have demonstrated performance that matches or even surpasses that of GNNs<sup>16</sup>.

Inspired by the work of Rampášek et al.<sup>17</sup>, this paper adopts a parallelized architecture, which sums the output of each GNN layer with that of the Transformer layer, and updates the features through multiple layers of iteration. Generally, GNNs can only aggregate messages from local neighbors, while Transformers can directly model the long-range dependencies between nodes through the attention mechanism. Combining the two in a parallel manner helps to balance the local and global features. For GNNs, this combination enables the aggregation of messages from distant nodes in each iteration, thus alleviating the problems of over-smoothing and over-squashing to a certain extent. Traditional GNNs exhibit limited expressive power and present opportunities for further enhancement in local information processing. However, standard Transformers typically fail to consider the incorporation of edge information, which is crucial for effective graph representation and understanding. This paper introduces enhancements to both components, and the main contributions are as follows:

- Edge-level positional encoding is introduced based on node-level random walk positional encoding. However, there may be scenarios where such encodings exist while the corresponding initial edge features are absent. To address this, two augmentation strategies are proposed to enhance the original graph, which are then used as inputs for GNNs and Transformers, respectively.
- GNNs typically only aggregate messages from direct neighbors, and this aggregation method limits their expressive power. To learn better local features, subgraphs centered around each node are extracted from the original graph, and each subgraph is encoded using GNNs. Since there may be the same nodes in different subgraphs, these encodings will be further integrated to obtain the final node representations.
- Edge features are incorporated into standard Transformers to enable more accurate modeling of node relationships during attention computation. Additionally, a dynamic fusion mechanism is designed to optimize the integration of outputs from both GNN layers and Transformer layers, harnessing the combined strengths of these models to enhance the overall output quality.
- In standard Transformer architectures, the complexity of the attention mechanism grows quadratically with the number of inputs. To address this challenge, a linear attention mechanism specifically tailored for graph data is introduced.

## Related work

Graph transformers (GTs) have emerged as a new type of architecture for the modeling of graph-structured data, attracting significant interest and attention. Existing GTs can be broadly classified into two categories: The first category focuses on integrating graph information into Transformers, enhancing their ability to understand and process graph data, allowing them to achieve superior performance in the graph domain. The second category aims to combine GNNs with Transformers, leveraging the strengths of both to achieve a more comprehensive analysis and modeling of graph data.

## Incorporating graph information into transformers

Transformers were initially designed for processing sequential data, but graph-structured data presents greater complexity. Directly applying Transformers to graph-structured data can result in information loss and reduced model performance. To address this, incorporating additional graph-based positional and structural information into transformers is crucial. Absolute encoding (AE) and relative encoding (RE) are two key techniques that effectively integrate this information into Transformers, enhancing their ability to understand and process graph data.

Absolute encoding is a relatively simple and effective encoding method widely used in models such as GNNs and Transformers. Common forms of absolute encoding include Laplacian eigenvectors and node degrees, which are added or concatenated to the initial node features to provide key information about the graph. Chen et al.<sup>18</sup> used eigenvectors of the graph Laplacian matrix to capture node structural information. However, when dealing

with directed graphs, Laplacian eigenvectors may include complex numbers, adding additional complexity and constraints. Hussain et al.<sup>19</sup> proposed a positional encoding based on Singular Value Decomposition (SVD) of the graph adjacency matrix, which can be applied to different types of graphs. Ying et al.<sup>20</sup> introduced a centrality encoding, which assigns two real-valued embedding vectors to each node based on its in-degree and out-degree, and adds these vectors to the node features as the input. This encoding captures the importance of nodes within the graph, further enriching the node features.

Many GTs also employ relative encoding. Unlike absolute encoding, which is applied to the initial node features only once, relative encoding dynamically adjusts node features based on the relative relationships between nodes<sup>13</sup>. This dynamic approach allows relative encoding to more accurately capture information and associations within the graph, addressing the limitations of absolute encoding, which compresses the graph structure into fixed-size vectors<sup>16</sup>. Kuang et al.<sup>21</sup> used Personalized PageRank (PPR) as a bias term for the attention weight matrix. This method effectively utilizes the graph's topology, allowing the model to focus more on neighbors that are more closely associated with the current node. Park et al.<sup>22</sup> introduced two sets of learnable positional encoding vectors: topological encoding and edge encoding, to represent relative positional relationships. When constructing the attention matrix, node features interact with these two encoding vectors, allowing for the integration of node-topology and node-edge interactions.

### Combining GNNs with transformers

The combination of GNNs and Transformers leverages the sensitivity of GNNs to local information and the ability of Transformers to handle global information and long-range dependencies, resulting in superior performance across various tasks. Based on the relative positions of GNN and Transformer blocks, their combination can be categorized into three types, as shown in Fig. 2: (a) stacking Transformer blocks on top of GNN blocks in a serial manner; (b) parallelizing GNN and Transformer blocks; (c) alternately stacking GNN and Transformer blocks.

Stacking Transformer blocks on top of GNN blocks is one of the most common integration methods. Mialon et al.<sup>23</sup> employ Graph Convolutional Kernel Networks (GCKNs) to enumerate the local sub-structures of each node, encode these sub-structures through kernel embeddings, and then perform aggregation. The initial node features and the outputs of the GCKNs are then concatenated and fed into the Transformer model. A similar two-stage strategy has been successful in computer vision, largely due to the high expressive power of CNNs on grids. However, unlike CNNs, GNNs are constrained by issues such as over-smoothing and over-squashing, which means that even with more expressive message-passing networks, some information may still be lost at early stages.

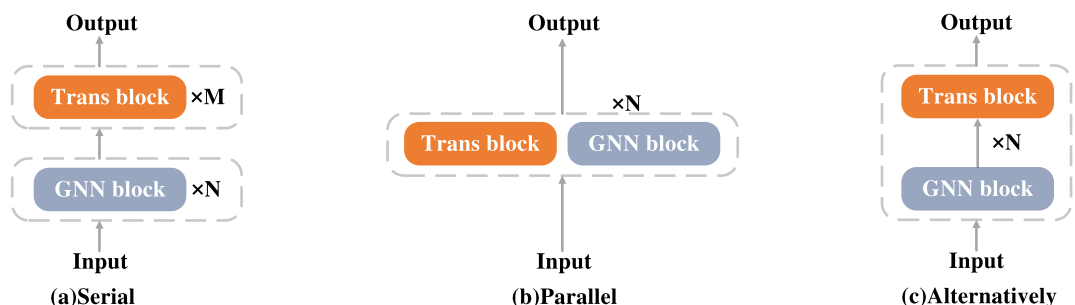
Parallelized architectures can mitigate some limitations of two-stage strategies, as the message passing in GNNs is no longer independent but influenced by Transformers. Rampásek et al.<sup>17</sup> proposed a general, powerful, and scalable architecture that updates the features by summing the outputs of local message-passing layers with those from global attention layers. However, this approach only incorporates edge features into the local message-passing layers. Performance could be further improved by integrating edge features into the global attention layers. The combination of convolution and self-attention in a Transformer encoder is helpful to improve representation learning. Lin et al.<sup>24</sup> adopted an alternating stacking strategy and proposed a graph-convolution-reinforced Transformer encoder to capture both local and global interactions for 3D human mesh reconstruction.

This paper combines the strengths of both GNNs and Transformers through a parallelized architecture that dynamically merges their outputs. In order to make better use of the edges, edge-level relative position encoding is introduced on the basis of node-level random walk position encoding to further enhance the edge features.

## Background

### Graph-structured data

A graph  $G$  is typically denoted as  $G = (V, E)$ , where  $V = \{v_1, v_2, \dots, v_n\}$  represents the set of nodes and  $E = \{e_1, e_2, \dots, e_m\}$  represents the set of edges. For each node  $v$ , its  $k$ -hop neighborhood is defined as  $N^k(v) = \{u \in V \mid d(v, u) \leq k\}$ . A general graph representation is a five-tuple:  $G(V, E, A, X, D)$ , where  $A \subseteq \mathbb{R}^{n \times n}$  denotes the adjacency matrix,  $X \subseteq \mathbb{R}^{n \times d}$  represents the feature matrix of the nodes,  $D \subseteq \mathbb{R}^{n \times n}$  is the degree matrix. Here,  $n$  and  $d$  denote the number of nodes and the dimensionality of the node features, respectively<sup>1</sup>.



**Fig. 2.** The combination of GNNs and transformers.

## Transformer architecture

The Transformer architecture, initially introduced by Google, leverages multi-head self-attention (MHA) and feedforward networks (FFNs) to build its encoder and decoder. Let  $X = [x_1, x_2, \dots, x_n] \subseteq \mathbb{R}^{n \times d}$  be the input to each Transformer layer, where  $x_i$  represents the feature of the  $i$ -th node. The computation of a single self-attention head is as follows:

$$\alpha = \frac{QK^T}{\sqrt{d_k}}, \quad \text{Attn}(X) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \quad (2)$$

In this context,  $W_Q$ ,  $W_K$ , and  $W_V$  are trainable parameters used to linearly transform the input  $X$  into the query matrix ( $Q$ ), key matrix ( $K$ ), and value matrix ( $V$ ).  $d_k$  represents the dimension of  $Q$  and  $K$ , and is used to scale the attention scores, thereby improving the stability of model training.

Multi-head self-attention is an effective strategy that enhances the model's ability to capture and represent information. It achieves this by computing self-attention for each head in parallel and then concatenating the outputs from all heads. The computation of multi-head self-attention is defined by the following formula:

$$MHA(X) = \left\|_{h=1}^H (\text{Attn}^{(h)}(X)) \right\| \quad (3)$$

$$\text{Attn}^{(h)}(X) = \text{softmax}\left(\frac{Q^{(h)}(K^{(h)})^T}{\sqrt{d_k}}\right)V^{(h)} \quad (4)$$

## Improved model

This paper introduces edge-level positional encoding in addition to the traditional node-level random walk positional encoding and proposes two methods to enhance the edges in the original graph. The enhanced graphs are then used as input to both GNNs and Transformers. For GNNs, message passing on subgraphs improves the model's ability to capture local information. For Transformers, the incorporation of edges enables the attention mechanism to more comprehensively account for the relationships between nodes. Finally, to better leverage the strengths of both GNNs and Transformers while balancing local and global features, the outputs of both models are dynamically fused. The overall architecture of the model is illustrated in Fig. 3.

## Random walk positional encoding

Initialize a random walk positional encoding  $P = [M^1, M^2, \dots, M^K]$ , and use this encoding to enhance the initial node and edge features. Here,  $K$  is a hyperparameter representing the maximum length of the random

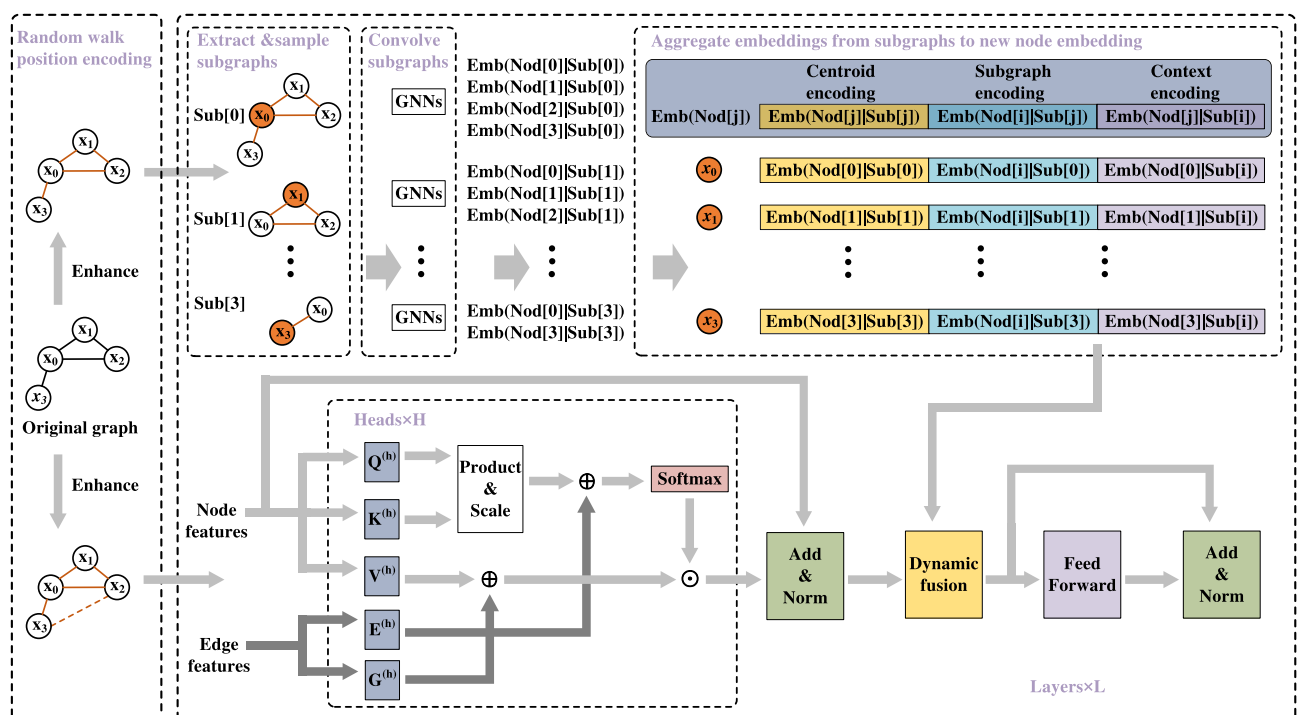


Fig. 3. Model overall architecture.

walk, and  $M = D^{-1}A$  is the probability transition matrix, where  $M_{ij}^k$  denotes the probability of node  $i$  transitioning to node  $j$  in  $k$  steps. The components of  $P$  contain significant structural information. For instance, if nodes  $i$  and  $j$  are direct neighbors, then  $M_{ij}$  is non-zero. In the absence of self-loops,  $M_{ii}^3$  is non-zero if and only if node  $i$  is part of a triangle, as shown in Fig. 4. In the Transformer architecture, position encoding based on random walk is more expressive than encoding based on the shortest path.

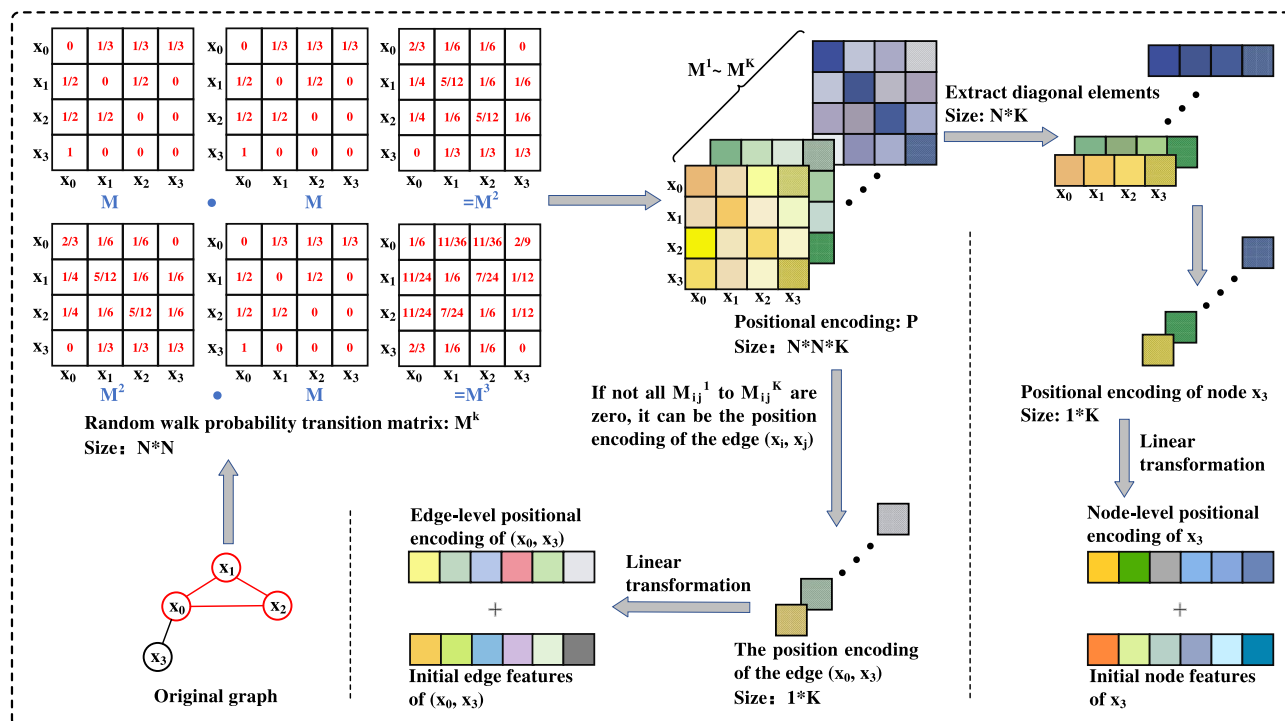
In previous work, node-level random walk encoding has been extensively studied and applied. For instance, Zhang et al.<sup>25</sup> and Dwivedi et al.<sup>26</sup> used the diagonal elements  $P_{ii}$  of the positional encoding matrix  $P$  to enhance node features. This method, compared to encoding based on the eigenvalues and eigenvectors of the graph Laplacian matrix, is invariant to signs and bases and has demonstrated strong performance. However, diagonal elements represent the probability of a node returning to itself, and relying solely on these diagonal elements may result in the loss of significant structural information. Non-diagonal elements reflect the relationships between nodes and even higher-order structures, so utilizing the complete encoding matrix  $P$  can enhance the edges.

Transformers, through the self-attention mechanism, can effectively model the correlations between different nodes in a fully connected graph without being constrained by the graph structure. However, when dealing with fully connected graphs or when a large number of additional edges are added, the performance of GNNs can be negatively impacted. This is due to the introduction of more noise from the increased number of node neighbors, which affects the effectiveness of message passing. Therefore, in scenarios where positional encoding is present but corresponding initial edge features are absent, different graphs are fed into GNNs and Transformers. Specifically, the graph input to GNNs is enhanced only for the edges that originally exist, while the graph input to Transformers is enhanced for both the originally existing edges and the relevant but non-existent edges.

### Subgraph-enhanced GNNs

For non-isomorphic graphs, even GNNs with fixed expressive power may struggle to differentiate between two original graphs. However, they might be able to differentiate between subgraphs of the original graph, as the expressive power required to distinguish between graphs scales with the size of the graphs<sup>27</sup>. By encoding local subgraphs of the original graph instead of the entire graph, GNNs generalize the aggregation patterns of traditional message-passing networks, allowing for the decomposition of difficult problems into simpler ones, thereby further enhancing the expressive capability of GNNs.

First, extract  $k$ -hop subgraphs centered on each node from the original graph, and apply GNNs to process each subgraph to obtain the node embeddings. The node embeddings learned in the subgraph will eventually be further transformed into the representations of the corresponding nodes in the original graph. Therefore, it is necessary to ensure that nodes and edges in the subgraph correspond to those in the original graph. The process of extracting  $k$ -hop subgraphs is shown in Algorithm 1. For nodes in the subgraph, the next-hop nodes are obtained by multiplying the previous-hop nodes with the adjacency matrix. As for the edges in the subgraph, they are identified based on edge index: if both the head and tail nodes of an edge are in the subgraph, then the edge belongs to the subgraph. Alternatively, this can be understood as follows: suppose the subgraph centered on  $x_2$  contains nodes  $x_0$  and  $x_1$ ; if there is an edge between these two nodes, then that edge is part of the subgraph.



**Fig. 4.** Random walk position encoding.

During subgraph extraction, the hop count of each node relative to the central node is obtained. These hop counts can serve as additional structural information, helping the model better understand the position and role of nodes within the subgraph. A smaller hop count indicates that the node is closer to the central node and has a stronger relationship with it, whereas a larger hop count means the node is farther from the central node with a more indirect connection. By encoding the hop counts and concatenating them with the initial node features of the subgraph, the model's performance can be further enhanced.

---

**Input:** the edge index of the original graph: `edge_index`, nodes in the original graph: `n`, hop count of a subgraph: `k`

**Output:** `nodes_mapper`, `edge_mapper`, hop count of each node relative to the central node in the subgraph: `hop_mapper`

---

```

1 create an adjacency matrix A based on the edge_index and the number of nodes n;
2 define an empty list hop_masks and add a boolean identity matrix as its first element;
  // the elements in hop_masks have a size of  $n \times n$ 
  // hop_masks[0] being True indicates the nodes in the 0-hop of the subgraph
3 for i in range(k) do
4   next_mask = A.matmul(hop_masks[i].float()) > 0; // calculate the next-hop nodes
5   append next_mask to hop_masks;
6   hop_mapper[(hop_mapper == - 1) & next_mask] = i + 1;
  // the hop count of a node in hop_mapper will not be updated more than once
7 end
8 nodes_mapper = (hop_mapper >= 0);
  // hop_mapper >= 0 indicates that the nodes are within the k-hop subgraph
9 edge_mapper = nodes_mapper[:, edge_index[0]] & nodes_mapper[:, edge_index[1]];
10 return nodes_mapper, edge_mapper, hop_mapper;

```

---

#### Algorithm 1. Extraction of a *k*-hop subgraph

---

In practice, to improve efficiency when processing multiple subgraphs from the original graph, all subgraphs are typically merged into a single large graph before being fed into GNNs. The merged graph preserves the independence of each subgraph, as there are no connections between them. Consequently, GNNs do not aggregate messages across different subgraphs during message passing. Since the same node may appear in different subgraphs, three types of encodings are used to integrate the node embeddings: centroid encoding, subgraph encoding, and context encoding. Centroid encoding focuses solely on the embedding of the central node in the subgraph; subgraph encoding sums up the embeddings of all nodes within the subgraph; and context encoding aggregates the embeddings of the same node across all subgraphs. Finally, these three encodings are fused to obtain the final representation of each node, as described by the following formula:

$$x_j^{\text{Centroid}} = \text{Emb}(\text{Nod}[j] \mid \text{Sub}[j]) \quad (5)$$

$$x_j^{\text{Subgraph}} = \text{Add} \left( \left\{ \text{Emb}(\text{Nod}[i] \mid \text{Sub}[j]) \mid i \in N^k(j) \right\} \right) \quad (6)$$

$$x_j^{\text{Context}} = \text{Add} \left( \left\{ \text{Emb}(\text{Nod}[j] \mid \text{Sub}[i]) \mid \forall i \text{ s.t. } j \in N^k(i) \right\} \right) \quad (7)$$

$$x_j = \text{Fuse} \left( x_j^{\text{Centroid}}, x_j^{\text{Subgraph}}, x_j^{\text{Context}} \right) \quad (8)$$

Here,  $\text{Emb}(\text{Nod}[j] \mid \text{Sub}[j])$  refers to the embedding of node *j* within subgraph *j*. If subgraphs are extracted for each node, the computational and storage overhead can become significant. To address this issue, a min-set-cover sampling method is employed to select a subset of subgraphs for processing. As shown in Algorithm 2, the min-set-cover sampling ensures that all nodes in the original graph are adequately covered while minimizing the number of selected subgraphs, thereby enhancing the quality and representativeness of sampling while maintaining efficiency. Additionally, the model samples subgraphs only during training, while using all subgraphs during evaluation, which can further enhance its generalization ability. Since the amount of data in the validation and test sets is usually much smaller than that in the training set, using all subgraphs during evaluation typically does not result in excessive memory consumption. If further reduction of the evaluation overhead is desired, it can be achieved by decreasing the batch size of the validation and test sets.



---

**Input:** nodes in each subgraph: nodes\_mapper, minimum redundancy: redundancy, node count in the original graph:  $n$

**Output:** sampled subgraph(central node): selected\_nodes, node coverage count after sampling: node\_covered\_times

```

1 randomly sample an initial subgraph;
2 for  $i$  in range( $n$ ) do
3     append newly sampled subgraph to selected_nodes;
4     update node_covered_times based on the latest sampled subgraph;
5     if node_covered_times.min()  $\geq$  redundancy then
6         break;
7     end
8     calculate the number of unused nodes in each subgraph;
9     // unused nodes:  $\sim((\text{node\_covered\_times} - \text{node\_covered\_times.min()}).\text{bool}())$ 
10    score each subgraph based on the number of unused nodes in the subgraph;
11    // assign a score of 0 to selected subgraphs to prevent resampling
12    select the subgraph with the highest score as the next subgraph to sample;
13 end
14 return selected_nodes, node_covered_times;

```

---

#### Algorithm 2. Min-set-cover subgraph sampling

---

After the subgraph sampling is completed, an important step is to update the nodes\_mapper, edge\_mapper, and hop\_mapper obtained during the subgraph extraction phase based on the sampled subgraphs (selected\_nodes). This update refers to filtering these mappers and retaining only the information relevant to the sampled subgraphs. If subgraph sampling is performed, only the central nodes in the sampled subgraphs will have the centroid encoding and subgraph encoding. To ensure that each node has all three types of encodings, the missing centroid and subgraph encodings need to be aligned. Before alignment, the hop counts of edges in the original graph relative to the central nodes must be calculated, as shown in Algorithm 3. First, starting from the central nodes of the sampled subgraphs, the one-hop edges directly connected to them are identified. Then, two-hop edges are found by starting from the tail nodes of the one-hop edges, and so on, until the hop counts for all edges in the original graph are determined. Finally, the centroid and subgraph encodings are aggregated based on the hop counts of the edges.

---

**Input:** edge\_index, selected\_nodes, node count in the original graph:  $n$

**Output:** hop counts of edges relative to the central nodes: edge\_hops

```

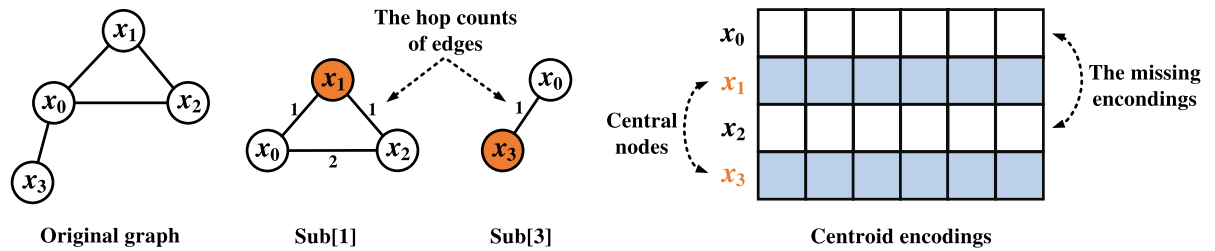
1 row, col = edge_index;
2 define two tensors, node_hops and edge_hops, and initialize their values to -1;
3 // the shapes of node_hops and edge_hops are (n,) and (row.shape,)
4 node_hops[selected_nodes] = 0;
5 selected_nodes = (node_hops == 0);
6 // in selected_nodes, central nodes are marked True, while others are False
7 initialize the hop count  $i = 1$ ;
8 for node_hops.min() < 0 do
9     // node_hops.min() < 0 indicates unvisited nodes,
10    // and the hops for their associated edges have not yet been calculated
11    source_near_edges = selected_nodes[row];
12    // in source_near_edges, edges with central nodes as head nodes are True
13    find the tail nodes corresponding to the edges with the central nodes as the head nodes: col[selected_nodes[row]];
14    update selected_nodes such that unvisited tail nodes are set to True, and all others are set to False;
15    // tail nodes become new central nodes for the next-hop edges
16    edge_hops[source_near_edges & selected_nodes[col]] =  $i$ ;
17    update the hop counts in node_hops to  $i$  for the nodes where selected_nodes is True;
18     $i = i + 1$ ;
19 end
20 return edge_hops;

```

---

#### Algorithm 3. The hop counts of edges relative to the central nodes

---



**Fig. 5.** The alignment of centroid encodings.

The alignment of centroid encodings is illustrated in Fig. 5. Two subgraphs centered on nodes  $x_1$  and  $x_3$  have been sampled, so only  $x_1$  and  $x_3$  possess centroid encoding. Furthermore, these subgraphs contain only 1-hop and 2-hop edges. First, aggregation is performed based on the tail nodes of the 1-hop edges ( $(x_1, x_0)$ ,  $(x_1, x_2)$ , and  $(x_3, x_0)$  are all 1-hop edges). The centroid encoding of  $x_0$  is obtained by aggregating the encodings of  $x_1$  and  $x_3$ . Since there is only one 1-hop edge with  $x_2$  as the tail node,  $x_2$  is aggregated only by  $x_1$ . Aggregation is then performed based on the 2-hop edges. The alignment of the subgraph encodings is similar to that of the centroid encoding.

### Edge-enhanced Transformers

Edge features are also crucial, as they provide key information for characterizing the relationships between nodes. In GNNs, a common practice is to incorporate edge features into the features of the associated nodes or to use edge features alongside node features during aggregation. However, these methods only propagate edge information to the nodes associated with these edges. To better leverage edge features, they can be integrated into Transformers, allowing these features to be considered during attention computation. In Transformers, there are mainly two ways to incorporate edge features: one is to use edge features to adjust the generation of the attention weight matrix, thereby affecting the attention distribution between nodes; the other is to add edge features to the value matrix ( $V$ ). By combining these two methods, the flexibility of the attention mechanism can be significantly improved. The improved attention computation formula is defined as follows:

$$E_{i,j} = \sigma(\rho((x_i W_Q + x_j W_K) \odot W_{Ew} e_{i,j}) + W_{Eb} e_{i,j}) \quad (9)$$

$$G_{i,j} = \sigma(\rho((x_i W_Q + x_j W_K) \odot W_{Gw} e_{i,j}) + W_{Gb} e_{i,j}) \quad (10)$$

$$\hat{\alpha}_{i,j} = \text{softmax}\left(\frac{(x_i W_Q)(x_j W_K)^T}{\sqrt{d_k}} + E_{i,j}\right) \quad (11)$$

$$\text{Attn}(x_i) = \sum_{j \in V} \hat{\alpha}_{i,j} (x_j W_V + G_{i,j}) \quad (12)$$

$$\rho(x) = (\text{ReLU}(x))^{1/2} - (\text{ReLU}(-x))^{1/2} \quad (13)$$

Here,  $\sigma(\cdot)$  denotes a nonlinear activation function, and  $\rho(\cdot)$  represents a signed square root function that aids in stabilizing the training process.  $e_{i,j}$  denotes the features of the edge connecting nodes  $i$  and  $j$ , while  $W_{Ew}$ ,  $W_{Eb}$ ,  $W_{Gw}$  and  $W_{Gb}$  are learnable weight matrices. This attention calculation method not only updates edges but also enables node aggregation influenced by edges. Compared to the scaled dot-product attention used in standard Transformers, this method captures the relative relationships between nodes in a more flexible and comprehensive manner.

In the specific implementation of Transformers, as described in the Background, self-attention is typically achieved through matrix multiplications among the query matrix ( $Q$ ), key matrix ( $K$ ), and value matrix ( $V$ ). The process entails the computation of attention weights and the aggregation of input features. However, the problem lies in the fact that the dimension of the attention matrix calculated in this way is related to the number of nodes, while the dimension of the edge features is not the same as that of the attention matrix. This results in the inability to effectively take the edges into account when calculating the attention.

Transformers can also be implemented through the way of message passing. First, map the query matrix ( $Q$ ), key matrix ( $K$ ), and value matrix ( $V$ ) according to the edge index. Then, calculate the attention weights based on  $Q$  and  $K$  and use them to weight  $V$ . Finally, aggregate the weighted results according to the edge index. These two approaches differ in implementation but are fundamentally the same. Since edge features need to be utilized in the attention calculation, the second approach is adopted. However, this method only calculates the attention between pairs of nodes that are directly connected. Therefore, to compute fully connected attention, it is necessary to provide a fully connected graph (connecting nodes that were not originally directly connected).

Combining GNNs and Transformers can leverage the advantages of both, resulting in superior performance across various tasks. Although it is common to stack Transformer blocks on top of GNN blocks in a serial manner, this two-stage approach can be limited by issues such as over-smoothing and over-squeezing in GNNs. To address these limitations, this model adopts a parallel structure. Specifically, the output of each GNN layer is summed with that of the Transformer layer and processed through multi-layer iterations to update the features.



Direct summation is a simple and intuitive fusion method that does not require additional parameters. However, this method treats the outputs of both models equally and lacks the ability to dynamically adjust their weights. In certain cases, it may lead to inappropriate weighting for specific tasks. To enhance the integration of outputs from both GNN and Transformer layers, a gating-based dynamic fusion mechanism is designed, which can adaptively adjust the fusion weights of the two. The detailed formulation is as follows:

$$\text{weights} = \sigma(W_{\text{gate}}[X_{\text{GNN}}, X_{\text{Trans}}] + W_{\text{bias}}) \quad (14)$$

$$X_{\text{fused}} = \text{weights} \odot X_{\text{GNN}} + (1 - \text{weights}) \odot X_{\text{Trans}} \quad (15)$$

### Reducing the complexity of Transformers

In Transformers, the attention mechanism can be modeled through an interaction graph, where an edge from node  $i$  to node  $j$  indicates that attention between them will be computed. The standard Transformer architecture employs a fully connected attention mechanism, meaning that each node can interact with all other nodes. However, this fully connected attention results in quadratic complexity relative to the number of nodes, and a large number of distant nodes can distract the target node from focusing on its local neighborhood. In the field of language modeling, some linear Transformers (such as BigBird<sup>28</sup> and Performer<sup>29</sup>) have made significant progress. However, due to the complexity of graph-structured data, designing suitable linear attention mechanisms becomes more challenging, and linear Transformers have not been extensively explored in the graph domain.

To reduce the complexity of Transformers while still incorporating edge features, this paper draws on the approach proposed by Shirzad et al.<sup>30</sup> Their linear attention consists of three components: local attention, expander graph attention, and global attention, as illustrated in Fig. 6. Local attention allows each node to focus on its first-order neighbors to model local interactions. Expander graph attention enables information to be propagated between distant nodes without requiring connections between all node pairs. Global attention is achieved by adding a small number of virtual nodes, each connected to all other nodes. Together, these three components form an interaction graph that, compared to a fully connected interaction graph, contains only a linear number of edges, thereby significantly reducing the computational cost of attention.

Expander graphs are a type of graph with special properties and are widely used in algorithms and theorems. An  $\epsilon$ -expander graph, commonly defined as a  $d$ -regular graph where each node has degree  $d$ , satisfies that the eigenvalues of its adjacency matrix meet the condition  $|\mu_i| \leq \epsilon d$  for all  $i \geq 2$  and  $\epsilon > 0$ . In a  $d$ -regular graph, the eigenvalues of the Laplacian matrix can be expressed as  $\lambda_i = d - \mu_i$ . Consequently,  $|d - \lambda_i| \leq \epsilon d$  for  $i \geq 2$ , indicating that all non-zero eigenvalues of the Laplacian matrix of an  $\epsilon$ -expander graph satisfy  $(1 - \epsilon)d \leq \lambda_i \leq (1 + \epsilon)d$ . Expander graphs are spectrally similar to fully connected graphs with the same number of nodes. Therefore, a linear attention based on expander graphs preserves the spectral properties of the fully connected attention mechanism. A graph  $G$  is an  $\epsilon$ -approximation of a graph  $H$ , which can be represented as:

$$(1 - \epsilon)H \preceq G \preceq (1 + \epsilon)H \quad (16)$$

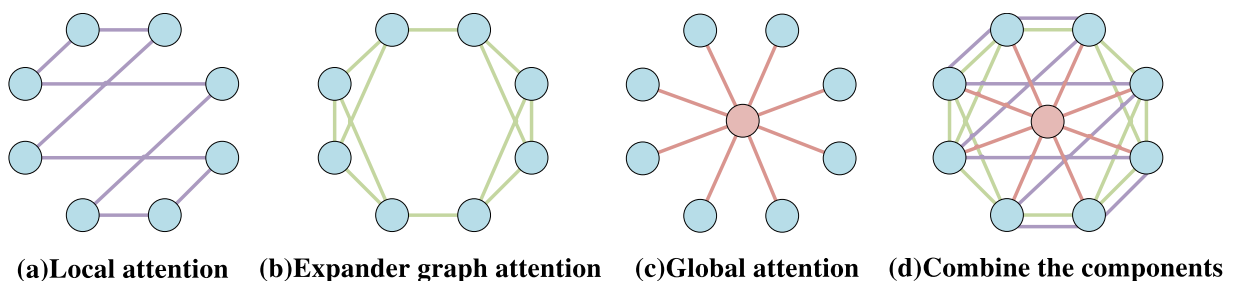
Here,  $H \preceq G$  indicates that for any vector  $x$ , the Laplacian matrices  $L_H$  and  $L_G$  satisfy the relationship  $x^T L_H x \leq x^T L_G x$ . A more detailed proof will follow.

Suppose graph  $G$  is a  $d$ -regular  $\epsilon$ -expander graph. For all vectors  $x$  orthogonal to the constant vector, inequality (17) holds. In the case of the fully connected graph  $K_n$ , Eq. (18) is satisfied for all such vectors  $x$ . If  $H = \frac{d}{n}K_n$  represents a fully connected graph with the same structure as  $K_n$  but with edge weights scaled by  $\frac{d}{n}$ , then Eq. (19) follows.

$$(1 - \epsilon)dx^T x \leq x^T L_G x \leq (1 + \epsilon)dx^T x \quad (17)$$

$$x^T L_{K_n} x = nx^T x \quad (18)$$

$$x^T L_H x = dx^T x \quad (19)$$



**Fig. 6.** Components of linear attention.

Here,  $n$  represents the number of nodes in the fully connected graph  $K_n$ . Substituting Eq. (19) into inequality (17) yields inequality (20), thus proving that graph  $G$  is an  $\epsilon$ -approximation of the fully connected graph  $H$ .

$$(1 - \epsilon)x^T L_H x \leq x^T L_G x \leq (1 + \epsilon)x^T L_H x \implies (1 - \epsilon)H \preceq G \preceq (1 + \epsilon)H \quad (20)$$

Here,  $H = \frac{d}{n}K_n$ . Substituting this into (20) yields inequality (21), demonstrating that graph  $G$  is also an approximation of the fully connected graph  $K_n$ .

$$\frac{(1 - \epsilon)d}{n}K_n \preceq G \preceq \frac{(1 + \epsilon)d}{n}K_n \quad (21)$$

Another property of expander graphs is that random walks mix well. Suppose  $v_0, v_1, v_2, \dots$  is a random walk sequence on a  $d$ -regular  $\epsilon$ -expander graph  $G$ , where  $v_0$  is a starting node chosen randomly according to the probability distribution  $\pi^{(0)}$ , and each subsequent node  $v_{t+1}$  is chosen randomly from among the  $d$  neighbors of  $v_t$ . It has been demonstrated that on a  $d$ -regular  $\epsilon$ -expander graph with  $n$  nodes, for any initial distribution  $\pi^{(0)}$ , where  $t = \Omega\left(\frac{\log(n/\delta)}{\epsilon}\right)$ , the distribution  $\pi^{(t)}$  satisfies the following inequality:

$$\left\| \pi^{(t)} - \frac{1}{n} \right\|_1 \leq \delta \quad (22)$$

Here,  $\delta > 0$ . This indicates that after a logarithmic number of steps, the initial probability distribution will converge to a uniform distribution. In the fully connected attention mechanism, it's clear that pairwise node interactions can be achieved within each Transformer layer. However, in the linear attention mechanism, certain node pairs are not directly connected, meaning a single Transformer layer cannot model interactions between all node pairs. Nevertheless, if this linear attention is built upon an  $\epsilon$ -expander graph, stacking at least  $t = \frac{\log(n/\delta)}{\epsilon}$  layers will allow for the modeling of pairwise interactions between most nodes.

A random  $d$ -regular graph, where the  $d$  neighbors of each node are chosen randomly, has a high probability of being an expander graph. Therefore, for simplicity, a random  $d$ -regular graph is used to construct the expander graph, as described in Algorithm 4. In fact, the first nontrivial eigenvalue  $\lambda_2$  of the Laplacian matrix of a random  $d$ -regular graph is closely related to  $\epsilon$  and the expansiveness of the graph. If the graph contains two edges separated by a distance of at least  $2k + 2$ , there exists an upper bound for  $\lambda_2$ , which plays a significant role in determining whether the graph is an expander graph:

$$\lambda_2 \leq d - 2\sqrt{d-1} + \frac{2\sqrt{d-1}-1}{k+1} \quad (23)$$

---

**Input:** original graph  $G = (V, E)$ , degree  $d$  of the regular graph

**Output:** edges of the expander graph

```

1 initialize the first nontrivial eigenvalue of the Laplacian matrix of the regular graph to  $\lambda = -1$ ;
2 while  $\lambda$  does not satisfy the properties of an expander graph do
    // generate the edges:(e_start, e_end) of a random d-regular graph
3   e_start = [0, 1, 2, ..., n] * d; // each node is repeated d times
4   e_end = [];
5   for i in range(d) do
6       rearrange [0, 1, 2, ..., n];
7       append the rearranged list to e_end;
8   end
9   compute the eigenvalue  $\lambda$  corresponding to the random d-regular graph based on the edges:(e_start, e_end)
10 end
11 return the edges of the expander graph;
```

---

**Algorithm 4.** Construction of the expander graph

---

## Experimental results and analysis

### Experimental setup and evaluation metrics

Software Environment: Windows Server 2019 Datacenter 64-bit operating system, PyCharm 2023.1.2 Community Edition, CUDA 11.8, Python 3.9, PyTorch 2.0.1. Hardware Environment: 2 Intel® Xeon® Gold 6154 @3.00GHz processors, 256GB DDR4 2666MHz memory, NVIDIA TITAN V with 12GB VRAM, 4TB mechanical hard drive, and 1TB solid-state drive.

Dataset	Graphs	Avg. nodes	Avg. edges	No. classes	Metric
ZINC-12K	12,000	23.1	49.8	1	MAE
CIFAR10	60,000	117.6	941.1	10	Accuracy
PATTERN	14,000	118.9	6078.6	2	Accuracy
CLUSTER	12,000	117.2	4301.7	6	Accuracy
MNIST	70,000	70.6	564.5	10	Accuracy
ogbg-molhiv	41,127	25.5	54.9	2	AUROC
ogbg-molpcba	437,929	26.0	56.2	128	AP
Peptides-func	15,535	150.9	307.3	10	AP
Peptides-struct	15,535	150.9	307.3	11	MAE

**Table 1.** Dataset statistics.

Hyperparameter	ZINC-12K	CIFAR10	PATTERN	CLUSTER	MNIST	ogbg-molhiv	ogbg-molpcba	Peptides-func	Peptides-struct
Transformer Layers	10	3	10	16	3	10	5	4	4
Hidden dim	64	52	64	48	52	64	384	96	96
Heads	8	4	8	4	4	4	4	4	8
Dropout	0	0	0	0.01	0	0.05	0.3	0	0.05
Attention dropout	0.2	0.5	0.2	0.5	0.5	0.5	0.5	0.5	0.5
Graph pooling	Sum	Mean	~	Mean	Mean	Mean	Mean	Mean	Mean
Epochs	2000	200	100	100	200	100	100	200	200

**Table 2.** Hyper-parameter settings for each dataset.

Different tasks have distinct characteristics and objectives, making it crucial to select appropriate evaluation metrics to accurately assess a model's performance. In this paper, the following evaluation metrics are used in the experiments:

1. MAE (Mean Absolute Error): A widely used metric for regression tasks, it measures the performance of a model by calculating the average absolute error between predicted values and actual values.
2. Accuracy: A common metric for classification tasks, it represents the proportion of correctly classified samples out of the total number of samples. Higher accuracy indicates better classification performance.
3. AUROC (Area Under the Receiver Operating Characteristic Curve): Mainly used to evaluate binary classification tasks, it represents the area under the ROC curve, with values ranging between 0 and 1.
4. AP (Average Precision): Commonly used in information retrieval and object detection tasks, it provides a comprehensive evaluation of the model's performance across different thresholds by considering both precision and recall.

### Experimental datasets and baseline models

To comprehensively evaluate the model's performance, five datasets were selected from Benchmarking GNNs<sup>31</sup>, and two datasets each were chosen from the Open Graph Benchmark<sup>32</sup> and the Long-Range Graph Benchmark<sup>33</sup> for comparison and ablation experiments. The statistical information for all datasets is provided in Table 1. These datasets cover a diverse range of graph tasks, including multi-class classification, multi-label classification, regression, and can be downloaded from the following links: <https://github.com/graphdeeplearning/benchmarking-gnns>, <https://ogb.stanford.edu>, and <https://github.com/vijaydwivedi75/lrgb>. In addition, the proposed model will be compared against several popular and state-of-the-art graph learning methods, including GNNs and GTs.

### Comparison experiments

First, the proposed method was evaluated on five datasets from Benchmarking GNNs<sup>31</sup>: ZINC, MNIST, CIFAR10, PATTERN, and CLUSTER. The hyper-parameters for each dataset are shown in Table 2, and the experimental results are shown in Table 3. The results clearly indicate that the proposed model significantly outperforms traditional message-passing networks and demonstrates superior performance compared to several existing Graph Transformers, with the most notable improvements observed on the CIFAR10 dataset.

Next, comparison experiments were conducted on two datasets each from Open Graph Benchmark<sup>32</sup> and Long-Range Graph Benchmark<sup>33</sup>. Table 4 shows the experimental results on the ogbg-molhiv and ogbg-molpcba datasets. Compared to other models, EHDGT shows a clear advantage, indicating its ability to learn more effective representations. As shown in Table 5, EHDGT performs excellently on both the Peptides-func and Peptides-struct datasets, particularly on the Peptides-func dataset, where its performance significantly exceeds that of other models, demonstrating strong long-range dependency capturing ability and generalization capability. Additionally, the paper compares EHDGT with a variant incorporating linear attention, named

Model	ZINC - 12K	MNIST	CIFAR10	PATTERN	CLUSTER
	MAE↓	Accuracy↑	Accuracy↑	Accuracy↑	Accuracy↑
GCN <sup>5</sup>	0.367 ± 0.011	90.705 ± 0.218	55.710 ± 0.381	71.892 ± 0.334	68.498 ± 0.976
GIN <sup>7</sup>	0.526 ± 0.051	96.485 ± 0.252	55.255 ± 1.527	85.387 ± 0.136	64.716 ± 1.553
GAT <sup>34</sup>	0.384 ± 0.007	95.535 ± 0.205	64.223 ± 0.455	78.271 ± 0.186	70.587 ± 0.447
PNA <sup>35</sup>	0.188 ± 0.004	97.94 ± 0.12	70.35 ± 0.63	–	–
GatedGCN <sup>36</sup>	0.282 ± 0.015	97.340 ± 0.143	67.312 ± 0.311	85.568 ± 0.088	73.840 ± 0.326
GatedGCN+GEANet <sup>37</sup>	0.218 ± 0.011	98.315 ± 0.097	73.857 ± 0.306	85.607 ± 0.038	77.013 ± 0.224
SAN <sup>38</sup>	0.139 ± 0.006	–	–	86.581 ± 0.037	76.691 ± 0.65
Graphormer <sup>20</sup>	0.122 ± 0.006	–	–	–	–
K-Subgraph SAT <sup>13</sup>	0.094 ± 0.008	–	–	86.848 ± 0.037	77.856 ± 0.104
EGT <sup>19</sup>	0.108 ± 0.009	98.173 ± 0.087	68.702 ± 0.409	86.821 ± 0.020	79.232 ± 0.348
Graphormer-GD <sup>39</sup>	0.081 ± 0.009	–	–	–	–
Cluster-GT <sup>40</sup>	0.071 ± 0.004	–	–	–	–
GraphGPS <sup>17</sup>	0.070 ± 0.004	98.051 ± 0.126	72.298 ± 0.356	86.685 ± 0.059	78.016 ± 0.180
LGI-GT <sup>41</sup>	0.069 ± 0.002	–	–	86.93 ± 0.04	78.19 ± 0.10
SubFormer-Spec <sup>42</sup>	0.068 ± 0.005	–	–	–	–
CSA-SPSE <sup>43</sup>	0.061 ± 0.003	98.269 ± 0.078	73.897 ± 0.524	87.064 ± 0.052	78.940 ± 0.132
TIGT <sup>44</sup>	<b>0.057 ± 0.002</b>	98.230 ± 0.133	73.955 ± 0.360	86.680 ± 0.056	78.033 ± 0.218
EHDGT (ours)	0.058 ± 0.002	<b>98.42 ± 0.12</b>	<b>75.34 ± 0.32</b>	<b>87.358 ± 0.065</b>	<b>79.526 ± 0.344</b>

**Table 3.** Test performance on five benchmarks from Benchmarking GNNs. Significant values are in bold.

Model	ogbg-molhiv	ogbg-molpcba
	AUROC↑	AP↑
GCN+virtual-node <sup>5</sup>	0.7599 ± 0.0119	0.2424 ± 0.0034
PNA <sup>35</sup>	0.7905 ± 0.0132	0.2838 ± 0.0035
DeeperGCN <sup>45</sup>	0.7858 ± 0.0117	0.2781 ± 0.0038
SAN <sup>38</sup>	0.7785 ± 0.0025	0.2765 ± 0.0042
GraphTrans <sup>46</sup>	–	0.2761 ± 0.0029
Exphormer <sup>30</sup>	0.7834 ± 0.0044	0.2849 ± 0.0025
GRIT <sup>47</sup>	0.7835 ± 0.0054	0.2362 ± 0.0020
GraphGPS <sup>17</sup>	0.7880 ± 0.0101	0.2907 ± 0.0028
HyPE-GT <sup>48</sup>	0.7893 ± 0.0005	–
EHDGT-L (ours)	0.7898 ± 0.0041	0.2912 ± 0.0025
EHDGT (ours)	<b>0.8013 ± 0.0069</b>	<b>0.2915 ± 0.0027</b>

**Table 4.** Test performance in graph-level OGB benchmarks. Significant values are in bold.

EHDGT-L. As shown in Tables 4 and 5, although EHDGT-L does not perform as well as EHDGT, it remains competitive.

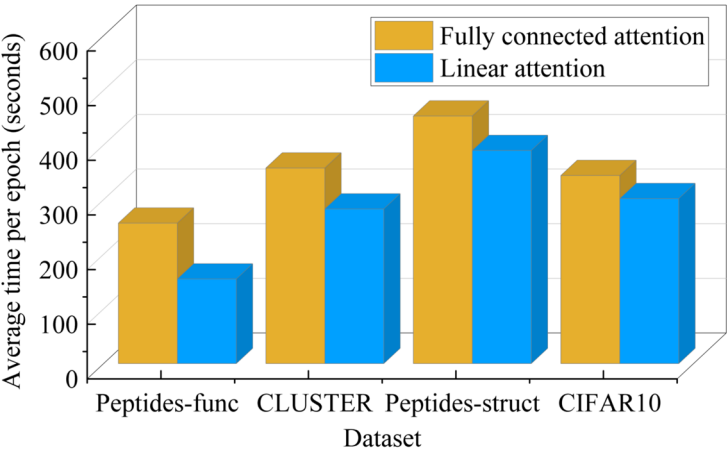
Since the linear attention is calculated based on the edges of the interaction graph, its computational complexity can be analyzed from the perspective of edges. The local attention is constructed based on the original graph, so it has  $O(|E|)$  interaction edges. In this paper, a random  $d$ -regular graph is used to construct the expander graph, so the expander graph attention introduces  $O(|V|)$  interaction edges. The global attention can be realized with only a small number of virtual nodes and contains only  $O(|V|)$  interaction edges. Since all components have a linear number of edges, the attention mechanism implemented based on them is also linear. As can be seen from Fig. 7, after using the linear attention, there is no need to calculate the attention between each pair of nodes, and the efficiency of the model is further improved.

### Ablation experiments

To assess the impact of each component in EHDGT on performance, ablation experiments were conducted across multiple datasets. Table 6 presents the experimental results for PATTERN. It is observed that both subgraph-enhanced GNNs and edge-enhanced Transformers, when combined with edge-level random walk positional encoding, improve the model's classification accuracy. Moreover, dynamically fusing the outputs of these two components leads to even more significant performance gains. Figure 8 illustrates the experimental results for MNIST and ZINC. It shows that even without subgraph-enhanced GNNs, the model outperforms

Model	Peptides-func	Peptides-struct
	AP↑	MAE↓
GCN <sup>5</sup>	0.5930 ± 0.0023	0.3496 ± 0.0013
GatedGCN <sup>36</sup>	0.5864 ± 0.0035	0.3420 ± 0.0013
GatedGCN+RWSE <sup>36</sup>	0.6069 ± 0.0035	0.3357 ± 0.0006
Transformer+LapPE <sup>14</sup>	0.6326 ± 0.0126	0.2529 ± 0.0016
SAN+LapPE <sup>38</sup>	0.6384 ± 0.0121	0.2683 ± 0.0043
SAN+RWSE <sup>38</sup>	0.6439 ± 0.0075	0.2545 ± 0.0012
GraphGPS <sup>17</sup>	0.6535 ± 0.0041	0.2500 ± 0.0012
Subgraphormer <sup>49</sup>	0.6415 ± 0.0052	0.2494 ± 0.0020
TIGT <sup>44</sup>	0.6679 ± 0.0074	<b>0.2485 ± 0.0015</b>
EHDGT-L (ours)	0.6620 ± 0.0025	0.2546 ± 0.0034
EHDGT (ours)	<b>0.6807 ± 0.0051</b>	0.2511 ± 0.0023

**Table 5.** Test performance on long-range graph benchmarks. Significant values are in bold.



**Fig. 7.** Average time per epoch.

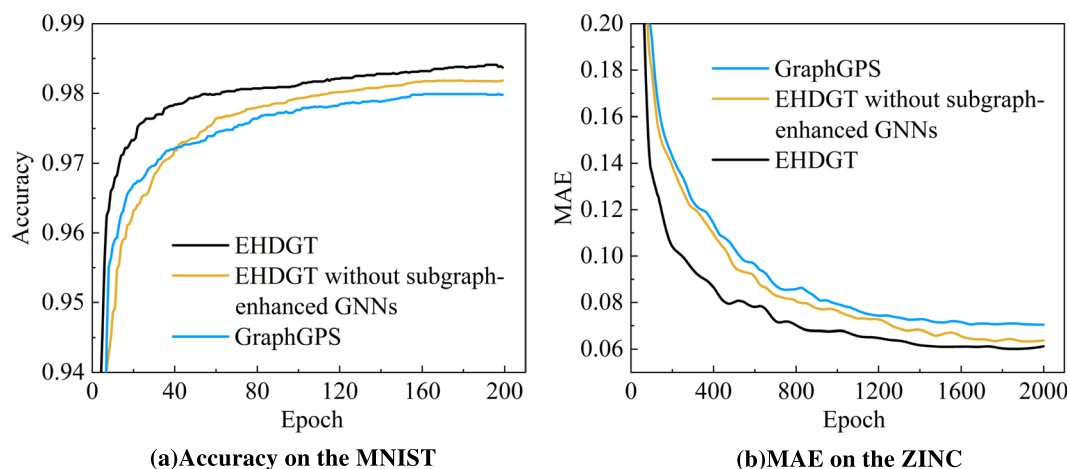
Edge-level positional encoding	Subgraph-enhanced GNNs	Integrating edge features into Transformers	Dynamical-fusion mechanism	Accuracy↑
–	–	–	–	86.765 ± 0.035
✓	✓	–	–	86.941 ± 0.042
✓	–	✓	–	87.246 ± 0.076
✓	✓	✓	–	87.329 ± 0.042
✓	✓	✓	✓	87.358 ± 0.065

**Table 6.** Ablation study on PATTERN.

GraphGPS. The inclusion of subgraph-enhanced GNNs not only accelerates convergence but also improves overall performance.

Application

The proposed model wants to predict latent but unannotated entity relations in the knowledge graph by learning high-quality representations of nodes and edges. This serves to complete the graph and enhance knowledge inference capabilities, providing richer and more accurate knowledge support for downstream tasks. By integrating GNN and Transformer into the graph learning process, the model leverages global self-attention mechanisms to capture high-order semantic relations across multiple hops by modeling interactions between arbitrary node pairs. To better preserve structural information and enhance the model's sensitivity to edge semantics, our approach incorporates edge information to enable edge feature prediction. For each edge in the graph, a positional vector is computed based on its structural position, and this encoding is embedded into the attention computation of the Transformer. This mechanism enables the model to distinguish edges with different



**Fig. 8.** Ablation study on MNIST and ZINC.

structural roles, while also improving its ability to model edge types and semantic weights, thereby enhancing the accuracy of relation prediction.

The model demonstrates significantly better performance on link prediction tasks compared to baseline methods that use either GNN or Transformer alone, especially in handling heterogeneous graphs and long-range semantic relations. This method effectively improves the connectivity and semantic richness of the wine industry knowledge graph, thereby markedly enhancing the quality and practical utility of the knowledge graph.

## Conclusions and prospect

This paper presents a graph representation learning model named EHDGT, which adopts a parallel architecture to dynamically fuse the outputs of GNNs and Transformers, thereby fully leveraging the strengths of both. Additionally, enhancements are made to each component: GNNs are used to encode subgraphs instead of the entire graph, further enhancing their ability to process local information; edge features are incorporated into the attention computation, enabling Transformers to more comprehensively model the relationships between nodes. To better utilize edge information, an edge-level positional encoding is introduced to enhance the original graph. EHDGT demonstrates strong performance and competitive advantages across multiple widely-used and representative datasets.

This work primarily evaluates the effectiveness of the proposed graph representation learning method through supervised downstream tasks, such as graph regression and graph classification. In the future, further exploration of Transformer-based unsupervised graph learning could be pursued, with potential applications in tasks like community detection and graph generation. While current Transformers effectively capture intra-graph attention among nodes within individual graph samples, they tend to overlook inter-graph correlations, which are critical for graph-level tasks. For example, in molecular graph data, molecules with similar structures often exhibit similar chemical properties and biological activities. A potential future direction is to introduce a set of learnable parameters that are independent of individual input graphs but shared across the entire dataset, enabling the model to implicitly capture relationships between different graph instances. Based on the proposed architecture, future work will focus on designing type-specific attention mechanisms for different node and edge types, to support knowledge graph completion and denoising.

Received: 14 February 2025; Accepted: 23 June 2025

Published online: 06 August 2025

## References

- Wu, B., Liang, X., Zhang, S. & Xu, R. Advances and applications in graph neural network. *Chin. J. Comput.* **45**, 35–68 (2022).
- Li, L., Liu, J., Ji, X., Wang, M. & Zhang, Z. Self-explainable graph transformer for link sign prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39, 12084–12092 (2025).
- Lei, H., Xu, J., Dong, X. & Ke, Y. Divergent paths: Separating homophilic and heterophilic learning for enhanced graph-level representations. arXiv preprint [arXiv:2504.05344](https://arxiv.org/abs/2504.05344) (2025).
- Wu, X. et al. Structure-enhanced contrastive learning for graph clustering. arXiv preprint [arXiv:2408.09790](https://arxiv.org/abs/2408.09790) (2024).
- Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks. arXiv preprint [arXiv:1609.02907](https://arxiv.org/abs/1609.02907) (2016).
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 1263–1272 (PMLR, 2017).
- Xu, K., Hu, W., Leskovec, J. & Jegelka, S. How powerful are graph neural networks? arXiv preprint [arXiv:1810.00826](https://arxiv.org/abs/1810.00826) (2018).
- Morris, C. et al. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 4602–4609 (2019).
- Li, Q., Han, Z. & Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32 (2018).
- Chen, D. et al. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 3438–3445 (2020).



11. Oono, K. & Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. arXiv preprint [arXiv:1905.10947](https://arxiv.org/abs/1905.10947) (2019).
12. Alon, U. & Yahav, E. On the bottleneck of graph neural networks and its practical implications. arXiv preprint [arXiv:2006.05205](https://arxiv.org/abs/2006.05205) (2020).
13. Chen, D., O'Bray, L. & Borgwardt, K. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, 3469–3489 (PMLR, 2022).
14. Vaswani, A. Attention is all you need. arXiv preprint [arXiv:1706.03762](https://arxiv.org/abs/1706.03762) (2017).
15. Qin, Z. et al. cosformer: Rethinking softmax in attention. arXiv preprint [arXiv:2202.08791](https://arxiv.org/abs/2202.08791) (2022).
16. Min, E. et al. Transformer for graphs: An overview from architecture perspective. arXiv preprint [arXiv:2202.08455](https://arxiv.org/abs/2202.08455) (2022).
17. Rampásek, L. et al. Recipe for a general, powerful, scalable graph transformer. *Adv. Neural. Inf. Process. Syst.* **35**, 14501–14515 (2022).
18. Chen, J., Gao, K., Li, G. & He, K. Nagphormer: A tokenized graph transformer for node classification in large graphs. arXiv preprint [arXiv:2206.04910](https://arxiv.org/abs/2206.04910) (2022).
19. Hussain, M. S., Zaki, M. J. & Subramanian, D. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 655–665 (2022).
20. Ying, C. et al. Do transformers really perform badly for graph representation?. *Adv. Neural. Inf. Process. Syst.* **34**, 28877–28888 (2021).
21. Kuang, W., WANG, Z., Li, Y., Wei, Z. & Ding, B. Coarformer: Transformer for large graph via graph coarsening, 2022. In <https://openreview.net/forum> (2021).
22. Park, W., Chang, W., Lee, D., Kim, J. & Hwang, S.-w. Grpe: Relative positional encoding for graph transformer. arXiv preprint [arXiv:2201.12787](https://arxiv.org/abs/2201.12787) (2022).
23. Mialon, G., Chen, D., Selosse, M. & Mairal, J. Graphit: Encoding graph structure in transformers. arXiv preprint [arXiv:2106.05667](https://arxiv.org/abs/2106.05667) (2021).
24. Lin, K., Wang, L. & Liu, Z. Mesh graphormer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 12939–12948 (2021).
25. Zhang, Z., Wang, M., Xiang, Y., Huang, Y. & Nehorai, A. Retgk: Graph kernels based on return probabilities of random walks. *Adv. Neural. Inf. Process. Syst.* **31**, 3964–3974 (2018).
26. Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y. & Bresson, X. Graph neural networks with learnable structural and positional representations. arXiv preprint [arXiv:2110.07875](https://arxiv.org/abs/2110.07875) (2021).
27. Loukas, A. How hard is to distinguish graphs with graph neural networks?. *Adv. Neural. Inf. Process. Syst.* **33**, 3465–3476 (2020).
28. Zaheer, M. et al. Big bird: Transformers for longer sequences. *Adv. Neural. Inf. Process. Syst.* **33**, 17283–17297 (2020).
29. Choromanski, K. et al. Rethinking attention with performers. arXiv preprint [arXiv:2009.14794](https://arxiv.org/abs/2009.14794) (2020).
30. Shirzad, H., Vellingker, A., Venkatachalam, B., Sutherland, D. J. & Sinop, A. K. Expformer: Sparse transformers for graphs. In *International Conference on Machine Learning*, 31613–31632 (PMLR, 2023).
31. Dwivedi, V. P. et al. Benchmarking graph neural networks. *J. Mach. Learn. Res.* **24**, 1–48 (2023).
32. Hu, W. et al. Open graph benchmark: Datasets for machine learning on graphs. *Adv. Neural. Inf. Process. Syst.* **33**, 22118–22133 (2020).
33. Dwivedi, V. P. et al. Long range graph benchmark. *Adv. Neural. Inf. Process. Syst.* **35**, 22326–22340 (2022).
34. Veličković, P. et al. Graph attention networks. arXiv preprint [arXiv:1710.10903](https://arxiv.org/abs/1710.10903) (2017).
35. Corso, G., Cavalleri, L., Beaini, D., Liò, P. & Veličković, P. Principal neighbourhood aggregation for graph nets. *Adv. Neural. Inf. Process. Syst.* **33**, 13260–13271 (2020).
36. Bresson, X. & Laurent, T. Residual gated graph convnets. arXiv preprint [arXiv:1711.07553](https://arxiv.org/abs/1711.07553) (2017).
37. Liang, J., Chen, M. & Liang, J. Graph external attention enhanced transformer. In *Proceedings of the 41st International Conference on Machine Learning, Proceedings of Machine Learning Research* (eds Salakhutdinov, R. et al.), Vol. 235 29560–29574 (PMLR, 2024).
38. Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V. & Tossou, P. Rethinking graph transformers with spectral attention. *Adv. Neural. Inf. Process. Syst.* **34**, 21618–21629 (2021).
39. Zhang, B., Luo, S., Wang, L. & He, D. Rethinking the expressive power of gnns via graph biconnectivity. arXiv preprint [arXiv:2301.09505](https://arxiv.org/abs/2301.09505) (2023).
40. Huang, S., Song, Y., Zhou, J. & Lin, Z. Cluster-wise graph transformer with dual-granularity kernelized attention. *Adv. Neural. Inf. Process. Syst.* **37**, 33376–33401 (2024).
41. Yin, S. & Zhong, G. Lgi-gt: Graph transformers with local and global operators interleaving. In *IJCAI*, 4504–4512 (2023).
42. Pengmei, Z. & Li, Z. Technical report: The graph spectral token-enhancing graph transformers with spectral information. arXiv preprint [arXiv:2404.05604](https://arxiv.org/abs/2404.05604) (2024).
43. Airale, L., Longa, A., Rigon, M., Passerini, A. & Passerone, R. Simple path structural encoding for graph transformers. arXiv preprint [arXiv:2502.09365](https://arxiv.org/abs/2502.09365) (2025).
44. Choi, Y. Y., Park, S. W., Lee, M. & Woo, Y. Topology-informed graph transformer. arXiv preprint [arXiv:2402.02005](https://arxiv.org/abs/2402.02005) (2024).
45. Li, G., Xiong, C., Thabet, A. & Ghanem, B. Deepergc: All you need to train deeper gcns. arXiv preprint [arXiv:2006.07739](https://arxiv.org/abs/2006.07739) (2020).
46. Wu, Z. et al. Representing long-range context for graph neural networks with global attention. *Adv. Neural. Inf. Process. Syst.* **34**, 13266–13279 (2021).
47. Ma, L. et al. Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, 23321–23337 (PMLR, 2023).
48. Bose, K. & Das, S. Hype-gt: where graph transformers meet hyperbolic positional encodings. arXiv preprint [arXiv:2312.06576](https://arxiv.org/abs/2312.06576) (2023).
49. Bar-Shalom, G., Bevilacqua, B. & Maron, H. Subgraphormer: Unifying subgraph gnns and graph transformers via graph products. arXiv preprint [arXiv:2402.08450](https://arxiv.org/abs/2402.08450) (2024).

## Funding

This paper is financially supported by Ningxia Key R&D Program (Key)Project (2023BDE02001) and Ningxia Natural Science Foundation Project (2023AAC03818).

## Additional information

**Correspondence** and requests for materials should be addressed to Q.Y.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025, corrected publication 2025