A Structure-Aware Higher-Order Message Passing Framework on Walk States for Graph Classification

Lu Bai[1], **Lin Du**[1], **Lixin Cui**[2,3]*, **Ming Li**[4], **Bo Jiang**[5], **Hangyuan Du**[6], **Ziyu Lyu**[7], **Xin Jin**[3]
[1]School of Artificial Intelligence, Beijing Normal University, Beijing, China.
[2] Engineering Research Center of State Financial Security, Ministry of Education, Central University of Finance and Economics, Beijing, China.
[3]School of Information, Central University of Finance and Economics, Beijing, China.
[4]Zhejiang Key Laboratory of Intelligent Education Technology and Application, Zhejiang Normal University, Jinhua, China.
[5]School of Computer Science and Technology, Anhui University, Hefei, China.
[6]School of Computer and Information Technology, Shanxi University, Taiyuan, China.
[7]School of Cyber Science and Technology, Sun Yat-Sen University, Shenzhen, China.
bailu@bnu.edu.cn, dulin@mail.bnu.edu.cn, cuilixin@cufe.edu.cn

Abstract

Standard Message Passing Neural Networks (MPNNs) are at most 1-dimensional Weisfeiler–Lehman (1-WL) expressive, which has spurred the development of higher-order models. In particular, many higher-order designs rely on extra global structural features (e.g., positional encodings or pairwise distances), whereas the k-WL hierarchy offers a principled and quantifiable path to increased expressiveness. However, directly simulating k-WL is computationally prohibitive, and existing k-WL variants often sacrifice expressiveness to gain efficiency, failing to capture critical structural distinctions. To overcome these shortcomings, we propose a higher-order message passing framework based on walk-induced lifted states with structure-aware state encoding. We prove that the default instantiation of our framework can separate certain 3-WL-indistinguishable graph pairs while avoiding dense tuple enumeration. On the challenging BREC expressiveness benchmark, our model achieves the highest total distinguishing accuracy among the compared higher-order GNN baselines. It also remains competitive on real-world graph classification tasks without using graph-wide positional or distance features.

1 Introduction

Graph classification is a fundamental problem with broad applications in chemistry, biology and beyond, for which Message Passing Neural Networks (MPNNs) have become the dominant learning paradigm due to their simplicity and scalability [20, 13, 36]. However, a well-known limitation is that standard MPNNs are expressively bounded, matching at most the 1-dimensional Weisfeiler–Lehman (1-WL) test in distinguishing non-isomorphic graphs [34, 24]. This limitation stems from the restricted node-level receptive field and the inability to effectively model interactions among neighbors, which can collapse distinct structures into identical node representations (e.g., graphs G and H in Fig. 1). Consequently, many challenging graph classification tasks remain hard for MPNNs, motivating the development of higher-order graph learning methods.

A key intuition is that lifting MPNNs to higher-order states (e.g., edges or walk-induced states) expands the receptive field and improves expressiveness. In Fig. 1 (and all subsequent figures), we

*Corresponding Author: Lixin Cui (cuilixin@cufe.edu.cn)

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

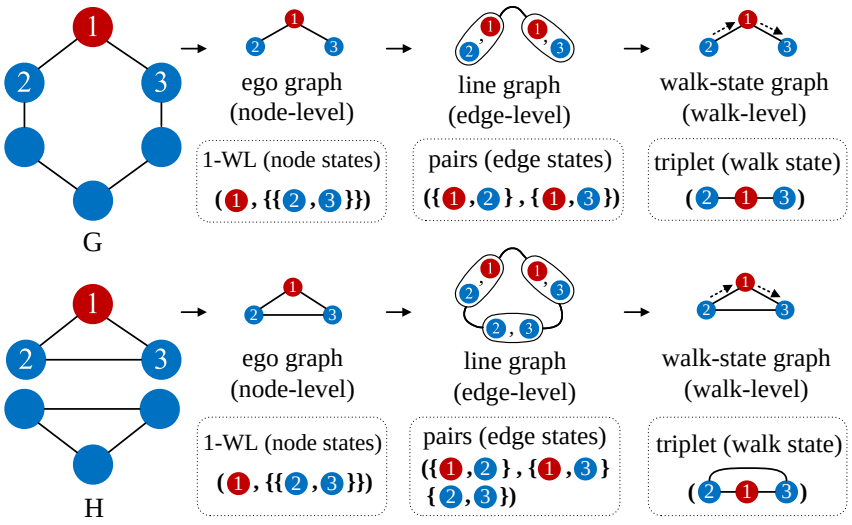


Figure 1: **Motivating example**. Lifting the modeling unit from nodes to edges and walks increases receptive field, making it easier to distinguish graphs G and H that 1-WL cannot separate.

assume identical initial node features; colors and node indices are used only for visualization, to indicate different roles/positions of the highlighted objects, and are not part of the input. Standard MPNNs operate on node states (ego graphs) and thus inherit the 1-WL limitation. A natural higher-order upgrade is to pass messages over edge states (node pairs) via the line graph, which can expose interactions between incident edges. However, edge-state representations can be indirect for capturing certain local higher-order patterns and may introduce redundant intermediate structures. This motivates a walk-induced state space that directly models fine-grained multi-node interactions, where each state is induced by a non-backtracking ℓ-hop walk. Note that for $\ell = 2$, it becomes a triplet state with the structure among the three nodes (Fig. 1).

To address the 1-WL limitation, a rich line of higher-order graph learning methods has been developed. Among them, the k-WL hierarchy [5, 30] offers a principled, quantifiable path to increased expressiveness as k grows, inspiring many k-WL-based GNNs for graph classification such as 1-2-3 GNN [24], PPGN [22], and δ-k-LGNN [26]. Meanwhile, many higher-order designs, including both k-WL-based and non-k-WL ones, further improve performance by using graph-dependent structural signals, such as positional encodings or pairwise distance features. For example, shortest-path distances (SPD) are used in I^2-GNN [19] and N^2-GNN [12]. While often effective, such signals may require relatively complex preprocessing, incurring nontrivial computational costs in practice.

Although k-WL-based approaches typically do not rely on extra global positional or distance signals, directly simulating k-WL remains computationally prohibitive due to the exponential growth of the k-tuple space, even for $k = 3$. As a result, existing k-WL-based models often face a trade-off between expressiveness and efficiency. This points to two central questions: *(i) what graph-native higher-order unit forms a tuple and thus admits direct k-WL replacement aggregation? (ii) how can we make such k-WL aggregation expressiveness-adaptive via controllable sparsification and localization?* A natural attempt to address these questions is to lift message passing from nodes to edges via the line graph transformation [9]; however, iterating the transformation is impractical, as iterated line graphs can grow rapidly for some graphs such as r-regular graphs with $r \geq 3$ [28].

Building upon the above observations, we introduce **a higher-order graph learning framework** based on walk-induced state lifting. The framework effectively models multi-node units while capturing complex graph structures. Specifically, we construct a lifted state graph whose nodes are non-backtracking ℓ-hop walk-induced states (triplets for $\ell = 2$). To ensure tractability, we define neighborhoods through single-position replacement in the spirit of k-FWL (i.e., the folklore WL), while restricting replacements to a controllable candidate set. In this way, we prune the exponential search space into a manageable, sparse structure. Overall, **the main contributions** are threefold.

- **A general walk-induced state lifting framework.** We propose a natural lift from node-level modeling to walk-induced ℓ-hop states, producing structure-aware representations on the induced

state graph that align well with k-WL/k-FWL-style refinements. With simple message passing and pooling, it yields higher-order graph-level representations without relying on graph-wide positional encodings, all-pairs distance features, or dense tuple enumeration.

- **Controllable sparsification and localization of FWL-style aggregation.** By restricting replacement operations to a configurable local candidate set rather than the entire graph, our approach provides a tunable balance between expressiveness and efficiency. The neighborhood scope can be adjusted on demand, allowing larger candidate sets to capture richer local structural asymmetries and separate hard regular graph pairs in practice.
- **Theoretical properties and strong empirical results.** We instantiate the framework as L2GNN ($\ell = 2$) and analyze its default sparsified and localized version. We show that it refines a symmetrized localized 3-FWL procedure, separates a standard 3-WL-indistinguishable strongly regular pair via candidate-indexed K_4 extensions, and scales with retained lifted states and candidate bundles rather than dense V^3 enumeration. Empirically, it achieves the highest total score among the compared baselines on the challenging BREC expressiveness benchmark and competitive performance on real-world graph classification tasks.

2 Preliminary concepts

2.1 The basic notation

Let $G = (V, E, X_V)$ be a graph with node set V, edge set E and node feature matrix $X_V = \{\mathbf{x}_v \in \mathbb{R}^d \mid v \in V\}$. We denote $n = |V|$ and $m = |E|$. For a node $v \in V$, its neighborhood is $\mathcal{N}(v) = \{u \in V \mid (v, u) \in E\}$. A pair of isomorphic graphs is denoted by $G \cong H$. We use $\{\{\cdot\}\}$ to denote a multiset and $[\ell] = \{1, 2, \ldots, \ell\}$.

Tuples and replacement. For an integer $k \geq 1$, an ordered k-tuple is $\mathbf{v} = (v_1, \ldots, v_k) \in V^k$. For $i \in \{1, \ldots, k\}$ and $w \in V$, we write $\mathbf{v}_{(i \to w)} = (v_1, \ldots, v_{i-1}, w, v_{i+1}, \ldots, v_k)$. We define the support of $\mathbf{v}$ as $\mathrm{supp}(\mathbf{v}) = \{v_1, \ldots, v_k\}$, i.e., the set of nodes appearing in $\mathbf{v}$, and $G[\mathrm{supp}(\mathbf{v})]$ denotes the induced subgraph.

2.2 The Weisfeiler–Lehman test

Here we briefly review the WL hierarchy. Given a labeled graph G, the **1-WL** iteratively refines node colors. At iteration $t > 0$, the node update rule is

$$c_v^{(t)} = \mathrm{HASH}\Big(c_v^{(t-1)},\ \{\{\, c_u^{(t-1)} \mid u \in \mathcal{N}(v) \,\}\}\Big), \tag{1}$$

where $\mathrm{HASH}(\cdot)$ is an injective function. For consistency and readability, we use a unified refinement notation for 1-WL, k-WL, and k-FWL, following the standard formulations in [29, 18].

k-WL. The k-WL colors k-tuples $\mathbf{v} = (v_1, \ldots, v_k) \in V^k$ by aggregating single-position replacements. Specifically, for each $i \in \{1, \ldots, k\}$,

$$c_{\mathbf{v},i}^{(t)} \leftarrow \{\{\, c_{\mathbf{v}_{(i \to w)}}^{(t-1)} \mid w \in V \,\}\}, \tag{2a}$$

$$c_{\mathbf{v}}^{(t)} = \mathrm{HASH}\Big(c_{\mathbf{v}}^{(t-1)},\ c_{\mathbf{v},1}^{(t)}, \ldots, c_{\mathbf{v},k}^{(t)}\Big). \tag{2b}$$

k-FWL. The folklore variant k-FWL differs from k-WL in that it first groups the k single-position replacements induced by each replacement node $w \in V$ into a k-dimensional color vector, and then aggregates the multiset of such vectors over w as below

$$c_{\mathbf{v},w}^{(t)} \leftarrow \Big(c_{\mathbf{v}_{(1 \to w)}}^{(t-1)}, \ldots, c_{\mathbf{v}_{(k \to w)}}^{(t-1)}\Big), \tag{3a}$$

$$c_{\mathbf{v}}^{(t)} = \mathrm{HASH}\Big(c_{\mathbf{v}}^{(t-1)},\ \{\{\, c_{\mathbf{v},w}^{(t)} \mid w \in V \,\}\}\Big). \tag{3b}$$

Note that, for any $k \geq 2$, k-FWL is as expressive as $(k{+}1)$-WL [16, 14, 15].

3 Related work

k-WL-based GNNs. k-WL-based methods aim to simulate the k-WL/k-FWL hierarchy to improve the expressiveness. 1-2-3 GNNs [24] directly use k-tuples to approximate the k-WL test, following

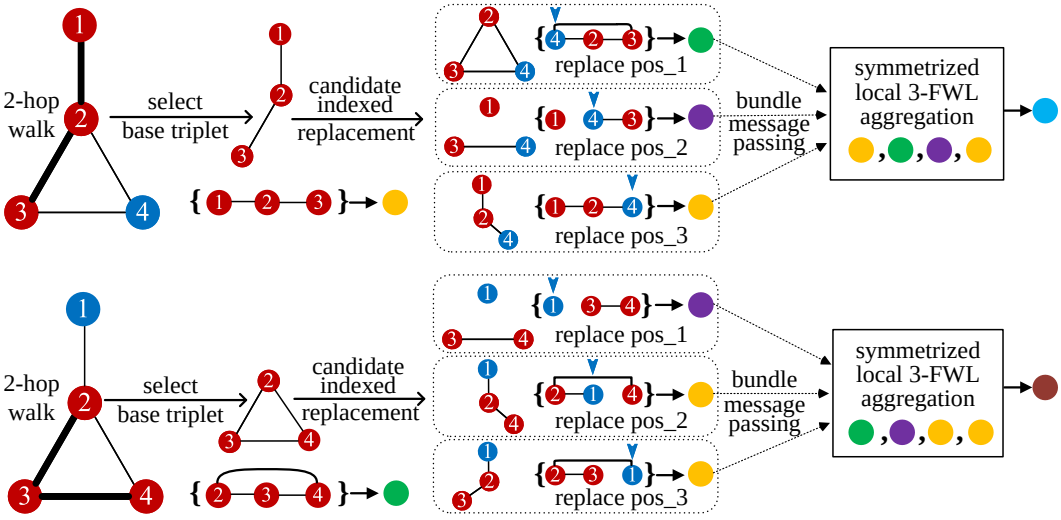


Figure 2: Overview of the default **L2GNN** (ℓ=2): walk-induced triplet lifting, candidate-indexed replacement, bundle message passing, and symmetrized local 3-FWL-style aggregation.

the iterative update rules in the spirit of standard k-WL. Taking a tensor-based approach, PPGN [22] employs matrix multiplication to achieve 3-WL expressive power. To mitigate the computational prohibitiveness of these global methods, δ-k-LGNNs [26] propose local k-WL variants that restrict the aggregation scope to local neighbors for better scalability. Similarly, KC-SetGNNs [39] relax ordered tuples to node sets of bounded size. Furthermore, recent extensions like the N^2-GNN framework [12] rethinks k-FWL by extending both the tuple-based aggregation and the neighborhood (aggregation scope) definition to enhance expressiveness. Our framework avoids enumerating the full k-tuple space; instead, we perform k-FWL-style aggregation on a constructed lifted state graph induced by short walks. Moreover, we design controllable sparsification and localization mechanisms to flexibly trade off efficiency and expressiveness.

Other higher-order GNNs. Beyond explicit k-WL/k-FWL simulations, higher-order expressiveness is often pursued by lifting message passing to structured local objects or by injecting additional structural encodings. Subgraph-based GNNs aggregate representations of policy-selected local subgraphs (e.g., KP-GNN [11], I^2-GNN [19]), often augmented with distance/positional encodings. While effective, such encodings often require graph-wide preprocessing and can add notable cost on large graphs. Substructure-based methods enhance MPNNs by injecting structural features that are hard to obtain from plain message passing, such as motif counting (e.g., GSN [3]). Transformer-based graph models (e.g., Graphormer [35]) also leverage global attention with structural biases. Compared with these approaches, our walk-induced state lifting method provides an explicit and structure-aware lifted graph that naturally aligns with information flow along walks. It is easy to construct without graph-wide preprocessing (e.g., positional encodings or all-pairs distances), and yields higher-order representations via message passing and pooling.

4 Method

4.1 Overview

We lift the input graph into a sparse walk-induced state graph, where each lifted node represents an ℓ-hop walk state. We instantiate $\ell = 2$ as **L2GNN**, with a default sparsified/localized model for experiments and an ordered global variant used only as a theoretical upper-bound reference because it requires dense triplets and global replacement candidates.

As illustrated in Fig. 2, our default L2GNN follows a sparse and localized workflow: (i) initializing 2-hop walk-induced triplet states and encoding each state with node features and a structure-aware token; (ii) constructing a sparse lifted state graph by connecting states via candidate-indexed single-position replacements under a controllable local candidate set; (iii) performing designed 3-FWL-style bundle

aggregation on the lifted graph to obtain state embeddings. Finally, after stacking multiple layers, we obtain a graph representation by applying a graph-level readout over the resulting embeddings.

4.2 The walk-state initialization and structure-aware state encoding

Walk states initialization. Let $G = (V, E, X_V)$ be an undirected graph. For an integer $\ell \geq 1$, we define the set of ℓ-hop walk states

$$\mathcal{T}_\ell^{seed}(G) = \left\{ \mathbf{v} = (v_1, \ldots, v_{\ell+1}) \in V^{\ell+1} \;\middle|\; \begin{array}{l} (v_i, v_{i+1}) \in E, \\ \forall i \in [\ell] \end{array} \right\}. \tag{4}$$

and enforce the non-backtracking constraint $v_i \neq v_{i+2}$ for all $i = 1, \ldots, \ell - 1$. In this paper we instantiate ℓ=2, where seed state is a triplet $\mathbf{v} = (v_1, v_2, v_3)$ induced by a 2-hop walk $v_1 \to v_2 \to v_3$.

Remark 4.1 (Canonicalized 2-hop triplet states and local closure *(implementation default)*). Although $\mathcal{T}_\ell^{\text{seed}}(G)$ is defined over ordered walks, our default implementation for $\ell = 2$ canonicalizes each 2-hop triplet by sorting node indices and removing duplicates:

$$\mathcal{T}_2^{\text{raw}}(G) = \left\{ (v_1, v_2, v_3) \;\middle|\; \begin{array}{l} (v_1, v_2) \in E,\ (v_2, v_3) \in E, \\ v_1 \neq v_3 \end{array} \right\}, \tag{5a}$$

$$\mathcal{T}_2^{seed}(G) = \{\text{can}(v_1, v_2, v_3) \mid (v_1, v_2, v_3) \in \mathcal{T}_2^{\text{raw}}(G)\}. \tag{5b}$$

where $\text{can}(v_1, v_2, v_3)$ sorts (v_1, v_2, v_3) into an unordered canonical triple. This canonicalization is used only to choose a representative of the unordered triplet state. For attributed graphs, the default encoder uses an invariant node-feature summary rather than an order-sensitive coordinate concatenation, so the representation does not depend on the arbitrary node-index order used to choose the representative. This substantially reduces redundancy on undirected graphs (e.g., opposite directions and permutations), while preserving the induced 3-node subgraph information that our model aims to encode. To support efficient replacement-style aggregation on sparse graphs, we further apply a lightweight local closure expansion: for each canonical triplet (v_1, v_2, v_3) and each s in a shared local candidate set, we add the three replaced canonical triplets $\text{can}(s, v_2, v_3)$, $\text{can}(v_1, s, v_3)$, and $\text{can}(v_1, v_2, s)$ into the lifted state set. We denote the resulting lifted triplet domain by $\mathcal{T}_2(G)$, which is seeded by 2-hop walks but may contain additional closure-expanded triplets.

Structure-aware state encoding. A key design goal is to make each lifted state aware of the internal relations among nodes in its support. We initialize each walk state $\mathbf{v} = (v_1, \ldots, v_{\ell+1})$ by combining (a) the node features of its coordinates and (b) a structure token extracted from the induced subgraph on its support. Ordered states may use coordinate concatenation. Our default model uses an invariant node-feature summary (the mean term is a scale-normalized copy of the sum for fixed ℓ):

$$\alpha_X(\mathbf{v}) = \left[\sum_{i=1}^{\ell+1} \mathbf{x}_{v_i} \,\Big\|\, \max_{i \in [\ell+1]} \mathbf{x}_{v_i} \,\Big\|\, \frac{1}{\ell+1} \sum_{i=1}^{\ell+1} \mathbf{x}_{v_i} \right], \tag{6}$$

where the max is taken coordinate-wise. The initial lifted-state representation is then

$$\mathbf{h}_{\mathbf{v}}^{(0)} = \text{ENC}(\alpha_X(\mathbf{v}),\ \phi(G[\text{supp}(\mathbf{v})])), \tag{7}$$

where ENC is an MLP encoder and $\phi(\cdot)$ summarizes the pairwise relations inside $\text{supp}(\mathbf{v})$.

Remark 4.2 (A concrete $\phi(\cdot)$ for ℓ=2 *(implementation default)*). For a triplet state $\mathbf{v} = (v_1, v_2, v_3)$, let $\mathcal{P} = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\}$. If edge attributes $\mathbf{e}_{a \to b}$ are available, we use an S_3-invariant pair descriptor $\boldsymbol{\psi}(\{a, b\}) = \mathbf{e}_{a \to b} + \mathbf{e}_{b \to a}$ (missing edges use $\mathbf{0}$), and aggregate over the three unordered pairs as

$$\phi(G[\{v_1, v_2, v_3\}]) = \Big[\sum_{\{a,b\} \in \mathcal{P}} \boldsymbol{\psi}(\{a, b\}) \,\Big\|\, \max_{\{a,b\} \in \mathcal{P}} \boldsymbol{\psi}(\{a, b\}) \Big], \tag{8}$$

where $\|$ denotes concatenation. If edge attributes are not given, we compute a 2-dim directed-existence bit for each unordered pair $\boldsymbol{\psi}(\{a, b\}) = [\mathbb{I}(a \to b \in E),\ \mathbb{I}(b \to a \in E)]$ and apply the same sum/max pooling across the three pairs.

4.3 The walk-state graph construction via position-aware replacements

The local candidate set. Directly emulating global k-FWL requires iterating over all $w \in V$, which is prohibitive. We therefore introduce a configurable local candidate set $\mathcal{S}(\mathbf{v}) \subseteq V$ to control sparsity

and locality. A typical choice is the union of neighborhoods of selected coordinates as

$$\mathcal{S}(\mathbf{v}) = \bigcup_{i \in \mathcal{I}} \mathcal{N}(v_i), \tag{9}$$

where $\mathcal{I} \subseteq \{1, \ldots, \ell+1\}$ is user-chosen (e.g., for ℓ=2, using all coordinates gives $\mathcal{S}(v_1, v_2, v_3) = \mathcal{N}(v_1) \cup \mathcal{N}(v_2) \cup \mathcal{N}(v_3)$ and this is our default choice).

Replacement edges with candidate-indexed bundles. We now construct a sparse lifted state graph $\mathcal{G}_\ell(G) = (\mathcal{T}_\ell(G), \mathcal{E}_\ell(G))$, where $\mathcal{T}_\ell(G)$ denotes the final lifted state domain used for message passing (for ℓ=2, this is $\mathcal{T}_2(G)$ after Remark 4.1). For each state $\mathbf{v} = (v_1, \ldots, v_{\ell+1})$, each position $i \in \{1, \ldots, \ell+1\}$, and each candidate node $s \in \mathcal{S}(\mathbf{v})$, define the replaced state $\mathbf{v}_{(i \to s)} = (v_1, \ldots, v_{i-1}, s, v_{i+1}, \ldots, v_{\ell+1})$. If $\mathbf{v}_{(i \to s)} \in \mathcal{T}_\ell(G)$, we add a directed edge

$$\mathbf{v}_{(i \to s)} \longrightarrow \mathbf{v} \quad \text{with type } i \text{ and candidate } s. \tag{10}$$

Every base state receives messages from its single-position replacement neighbors. In our aggregation (Section 4.4), we group incoming replacement neighbors by the candidate node s to form a replacement bundle, where position type i is optional and mainly used in the ordered instantiation.

Remark 4.3 (Local closure expansion and complete-bundle filtering *(implementation default, ℓ=2)*). On sparse walk-induced triplets, a single-position replacement may fall outside the current state set. Our default implementation therefore performs one step of local closure expansion: for each triplet (v_1, v_2, v_3) and each s in a shared local candidate set $S(v_1, v_2, v_3) = \mathcal{N}(v_1) \cup \mathcal{N}(v_2) \cup \mathcal{N}(v_3)$, we add all three replaced triplets (s, v_2, v_3), (v_1, s, v_3), and (v_1, v_2, s) into the lifted state set (with canonicalization when enabled). After expansion, we further restrict candidate nodes to those producing a complete replacement bundle, i.e., we only keep s such that all three replaced triplets exist in the lifted set. This yields clean per-$(\mathbf{v}, s)$ replacement bundles and stabilizes the aggregations.

4.4 The k-FWL-style aggregation on walk states

Let $\mathbf{h}_{\mathbf{v}}^{(p)} \in \mathbb{R}^d$ denote the representation of lifted state $\mathbf{v} \in \mathcal{T}_\ell(G)$ at layer p. Our update follows the two-stage pattern of k-FWL, but replaces the global domain V by the localized candidate set $\mathcal{S}(\mathbf{v})$. In our default implementation we further restrict aggregation to the complete candidate set defined as

$$\mathcal{S}^\star(\mathbf{v}) = \left\{ s \in \mathcal{S}(\mathbf{v}) \;\middle|\; \mathbf{v}_{(i \to s)} \in \mathcal{T}_\ell(G) \text{ for all } i \in [\ell{+}1] \right\}. \tag{11}$$

Stage I: bundle aggregation for each candidate.
For each $\mathbf{v}$ and each $s \in \mathcal{S}^\star(\mathbf{v})$, we collect the (size ℓ+1) replacement bundle as

$$\mathcal{B}^{(p)}(\mathbf{v}, s) = \{\!\{ \mathbf{h}^{(p)}_{\mathbf{v}_{(i \to s)}} \mid i \in [\ell{+}1] \}\!\}. \tag{12}$$

Our default instantiation uses an $S_{\ell+1}$-invariant DeepSets-style [37] aggregator over the bundle. For ℓ=2, the candidate-wise aggregated message is

$$\mathbf{g}^{(p)}_{\mathbf{v},s} = \sum_{i=1}^{3} \rho\Big(\mathbf{h}^{(p)}_{\mathbf{v}_{(i \to s)}}\Big), \tag{13}$$

where ρ is a shared learnable map that treats the three single-position replacements as an unordered set. In the ordered instantiation, one may alternatively use position-sensitive/type-specific maps to preserve coordinate roles.

Stage II: aggregate over candidates and update.
We aggregate the bundle embeddings across candidates in the local set as

$$\mathbf{m}^{(p)}_{\mathbf{v}} = \mathrm{AGGR}_s\Big(\{\!\{ \mathbf{g}^{(p)}_{\mathbf{v},s} \mid s \in \mathcal{S}^\star(\mathbf{v}) \}\!\}\Big), \tag{14}$$

and update the lifted-state representation via an injective MLP with a GIN-style [34] residual as

$$\mathbf{h}^{(p+1)}_{\mathbf{v}} = \mathrm{UPD}\Big((1+\epsilon)\mathbf{h}^{(p)}_{\mathbf{v}} \,\Big\|\, \mathbf{m}^{(p)}_{\mathbf{v}}\Big), \tag{15}$$

where UPD denotes the update function, ϵ is learnable and $\|$ denotes concatenation. If $\mathcal{S}^\star(\mathbf{v}) = \emptyset$, we set $\mathbf{m}^{(p)}_{\mathbf{v}} = \mathbf{0}$, so the layer gracefully reduces to a residual MLP update.

4.5 The multi-layer architecture and graph-level readout

The multi-layer stacking. We stack P localized k-FWL-style layers to obtain deep lifted-state representations

$$\begin{aligned} \mathbf{H}^{(0)} &= \{\mathbf{h}_{\mathbf{v}}^{(0)}\}_{\mathbf{v}\in\mathcal{T}_\ell(G)}, \\ \mathbf{H}^{(p+1)} &= \mathrm{Layer}\left(\mathbf{H}^{(p)};\, \mathcal{G}_\ell(G)\right), \quad p = 0, \dots, P-1. \end{aligned} \tag{16}$$

The skip-cat and pooling. Following the skip-cat [33] strategy, we concatenate representations from all layers before readout as

$$\mathbf{h}_{\mathbf{v}}^{\mathrm{skip}} = \mathbf{h}_{\mathbf{v}}^{(1)} \,\|\, \cdots \,\|\, \mathbf{h}_{\mathbf{v}}^{(P)}. \tag{17}$$

We then compute a graph-level representation by pooling over lifted states as

$$\mathbf{h}_G = \mathrm{READOUT}\Big(\{\!\{\, \mathbf{h}_{\mathbf{v}}^{\mathrm{skip}} \mid \mathbf{v} \in \mathcal{T}_\ell(G) \,\}\!\}\Big), \tag{18}$$

where READOUT can be sum/mean/max, Set2Set, or a combination such as $\mathrm{READOUT} = [\mathrm{SUM}\|\mathrm{MAX}]$. Finally, an MLP classifier maps $\mathbf{h}_G$ to class logits.

5 Theoretical properties

5.1 The expressive power of L2GNN

For theory, let $\mathcal{U}(G) \subseteq V^3$ be a triplet domain closed under the selected local replacements. For every $\mathbf{v} = (v_1, v_2, v_3) \in \mathcal{U}(G)$, position $i \in \{1, 2, 3\}$, and $s \in \mathcal{S}(\mathbf{v})$, the replacement $\mathbf{v}_{(i\to s)}$ belongs to $\mathcal{U}(G)$. When canonicalization is used, this replacement is understood as applying the replacement first and then mapping the resulting triplet to its canonical representative. In the default implementation, this closure is approximated by one-step local closure and complete-bundle filtering. We leave all formal proofs and the detailed complexity analysis to Appendix A.

Symmetrized localized 3-FWL. The default L2GNN used in our experiments is not a dense global 3-FWL implementation. It is a sparsified, symmetrized, and localized 3-FWL-style refinement.

Definition 5.1 (Symmetrized localized 3-FWL)**.** Let $\mathcal{S}^\star(\mathbf{v})$ be the complete local candidate set. For a triplet $\mathbf{v} = (v_1, v_2, v_3) \in \mathcal{U}(G)$, define

$$\sigma_{\mathbf{v},s}^{(p+1)} = \{\!\{c_{\mathbf{v}_{(1\to s)}}^{(p)}, c_{\mathbf{v}_{(2\to s)}}^{(p)}, c_{\mathbf{v}_{(3\to s)}}^{(p)}\}\!\}, \qquad s \in \mathcal{S}^\star(\mathbf{v}), \tag{19a}$$

$$c_{\mathbf{v}}^{(p+1)} = \mathrm{HASH}\Big(c_{\mathbf{v}}^{(p)}, \{\!\{\sigma_{\mathbf{v},s}^{(p+1)} \mid s \in \mathcal{S}^\star(\mathbf{v})\}\!\}\Big), \tag{19b}$$

where HASH is injective. In default L2GNN, $\mathcal{S}(\mathbf{v}) = \mathcal{N}(v_1) \cup \mathcal{N}(v_2) \cup \mathcal{N}(v_3)$.

Proposition 5.2 (Default L2GNN refines symmetrized localized 3-FWL)**.** *Assume injective initial triplet encoding, injective multiset aggregation inside each replacement bundle, injective multiset aggregation over candidates, and injective state updates. Then each layer of default L2GNN refines at least as finely as Definition 5.1.*

Proposition 5.3 (Relation to 3-WL, 3-FWL, and degeneration)**.** *The following statements hold.*

1. *If $\mathcal{U}(G) = V^3$, $\mathcal{S}(\mathbf{v}) = V$, and the bundle combiner is position-sensitive, L2GNN simulates ordered global 3-FWL, hence is as expressive as 4-WL and strictly separates some pairs not separated by 3-WL.*
2. *Default L2GNN can be viewed as an operational localized and symmetrized restriction of global 3-FWL, as it replaces the global candidate set V by $\mathcal{S}^\star(\mathbf{v})$ and replaces the ordered replacement vector by an unordered replacement multiset.*
3. *If $\mathcal{S}^\star(\mathbf{v}) = \emptyset$, the layer degenerates to a residual MLP update and performs no new refinement.*
4. *If $\mathcal{S}(\mathbf{v}) \subseteq \mathrm{supp}(\mathbf{v})$, no external witness node is queried.*

Proposition 5.4 (K_4-relevant local four-node extension exposed by L2GNN)**.** *Let $\mathbf{v} = (v_1, v_2, v_3)$ be a triplet state and let $s \in S^\star(\mathbf{v}) \setminus \mathrm{supp}(\mathbf{v})$ be an external candidate. If the initial triplet color injectively encodes induced three-node isomorphism types, then the base triplet structure together with the replacement bundle $\mathcal{B}_G(\mathbf{v}, s)$ determines whether the four-node extension $G[\{v_1, v_2, v_3, s\}]$ is a K_4 whenever $G[\{v_1, v_2, v_3\}] \cong K_3$.*

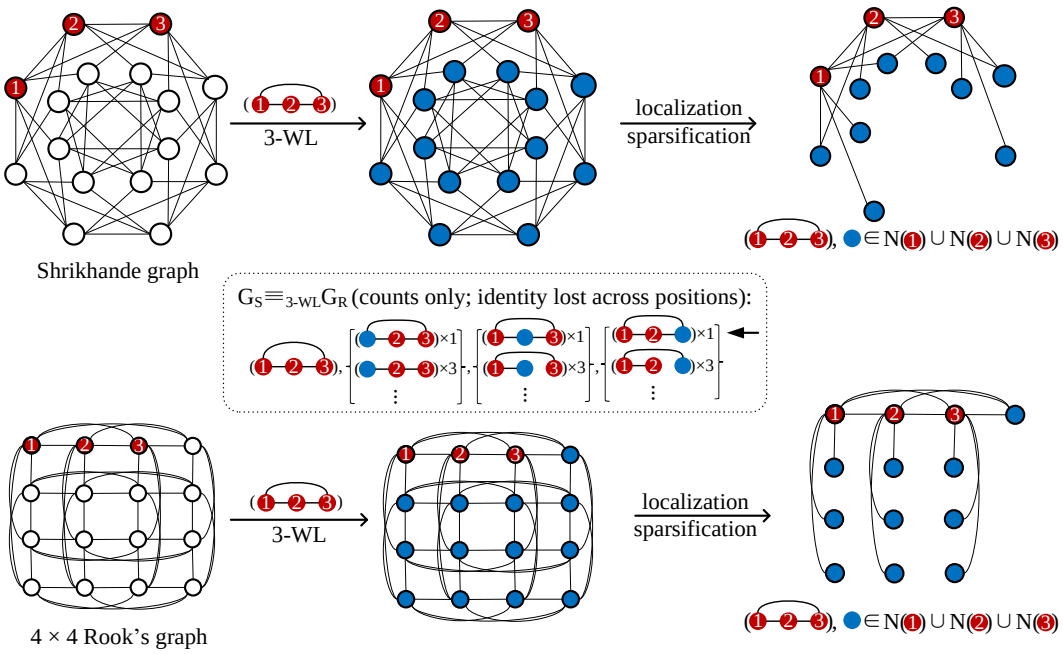


Figure 3: Shrikhande and 4×4 Rook's graphs. 3-WL fails, whereas L2GNN successfully distinguishes them through effective sparsification and localization designs.

Proposition 5.5 (Default L2GNN is not upper bounded by 3-WL)**.** *Consider simple undirected graphs and the default candidate set $\mathcal{S}(\mathbf{v}) = \mathcal{N}(v_1) \cup \mathcal{N}(v_2) \cup \mathcal{N}(v_3)$ with one-step local closure and complete-bundle filtering. Under the same injectivity assumptions as Proposition 5.2 and an injective graph-level readout, one L2GNN layer detects candidate-indexed K_4 extensions. For any triangular triplet $\mathbf{v} = (v_1, v_2, v_3)$ and external candidate $s \in \mathcal{S}^\star(\mathbf{v}) \setminus \mathrm{supp}(\mathbf{v})$, its candidate bundle determines whether $G[\{v_1, v_2, v_3, s\}] \cong K_4$.*

Consequently, default L2GNN separates any two graphs with different numbers of K_4 subgraphs. In particular, it distinguishes the 4×4 Rook's graph and the Shrikhande graph (see Fig. 3), while this pair is not separated by 3-WL under the convention used in this paper [1].

5.2 The complexity of the default L2GNN

With retained triplets $T = |\mathcal{T}_2(G)|$ and maximum candidate count $K = \max_{\mathbf{v}} |\mathcal{S}^\star(\mathbf{v})|$, each layer of L2GNN only costs $O(KT)$ time and $O(T + KT)$ memory, avoiding the $\Theta(n^3)$ states and $\Theta(n^4)$ per-iteration cost of global 3-WL/3-FWL. The detailed bounds are in Appendix A.6.

6 Experiments

We evaluate the default L2GNN on the challenging expressiveness-oriented BREC benchmark [32] and real-world graph classification tasks. Additional experiments, implementation details, ablations, and code are provided in the Appendix and supplementary material.

6.1 Expressiveness benchmark

Dataset. To verify the expressive power of our model, we evaluate **L2GNN** on **BREC** [32], a benchmark that has been used in recent work to assess the realized expressiveness of beyond-1-WL GNNs. BREC contains harder non-isomorphic pairs than earlier expressiveness datasets. It consists of four groups of graph pairs (Basic/Regular/Extension/CFI), spanning difficulties from 1-WL-indistinguishable up to pairs that remain indistinguishable even by 4-WL. In total, BREC contains 400 pairs (800 distinct graphs) with heterogeneous graph sizes.

Table 1: Pair distinguishing accuracies on BREC. Num. denotes the number of distinguished pairs and Acc. denotes accuracy. Best results are in bold; second and third best are underlined.

Model	Basic (60)		Regular (140)		Extension (100)		CFI (100)		Total (400)	
	Num.	Acc.	Num.	Acc.	Num.	Acc.	Num.	Acc.	Num.	Acc.
Non-GNN										
3-WL	60	100%	50	35.7%	100	100%	60	60.0%	270	67.5%
Subgraph-based										
KP-GNN (2022)	60	100%	106	75.7%	98	98%	11	11%	275	68.8%
I^2-GNN (2023)	60	100%	100	71.4%	100	100%	21	21%	<u>281</u>	<u>70.2%</u>
Substructure-based										
GSN (2022)	60	100%	99	70.7%	95	95%	0	0%	254	63.5%
Transformer-based										
Graphormer (2021)	16	26.7%	12	8.6%	41	41%	10	10%	79	19.8%
Cell-complex-based										
CIN (2021)	59	98.3%	135	96.4%	85	85%	0	0%	279	69.8%
k-WL-based										
PPGN (2019)	60	100%	50	35.7%	100	100%	23	23%	233	58.2%
δ-k-LGNN (2020)	60	100%	50	35.7%	100	100%	6	6%	216	54%
KC-SetGNN (2022)	60	100%	50	35.7%	100	100%	1	1%	211	52.8%
N^2-GNN (2023)	60	100%	100	71.4%	100	100%	27	27%	<u>287</u>	<u>71.8%</u>
CoSp-PPGN (2025)	60	100%	50	35.7%	100	100%	33	33%	243	60.8%
L2GNN (Ours)	60	100%	137	97.9%	100	100%	0	0%	**297**	**74.3%**

Models. We compare with strong baselines covering major beyond-1-WL families: a non-GNN baseline (3-WL); subgraph-based GNNs (KP-GNN [11], I^2-GNN [19] with SPD); a substructure-counting GNN (GSN [3]); a transformer-based model (Graphormer [35]); a cell-complex model (CIN [2]); and k-WL-based GNNs (PPGN [22], δ-k-LGNN [26], KC-SetGNN [39], N^2-GNN [12] with SPD, CoSp-PPGN [7]). Performance is reported following the standard BREC protocol.

Results. Table 1 reports pair distinguishing accuracies on BREC. Without using graph-wide positional encodings or all-pairs shortest-path distances in the default model, **L2GNN** achieves the highest total score among the compared baselines in Table 1, distinguishing $297/400$ pairs (**74.3%**). L2GNN improves over the strongest baseline in the table, N^2-GNN ($287/400$, 71.8%). This improvement mainly comes from the Regular split, where L2GNN distinguishes $137/140$ pairs.

The CFI split limitation and the optional SPD-profile extension. CFI remains a known bottleneck for sparse/local higher-order methods without graph-wide structural features or dense/global k-WL-style simulations [32]. This is consistent with our localization and sparsification design, where the restricted candidate set and hop budget can limit performance on larger CFI graphs. As a diagnostic extension, **L2GNN + SPD-profile** adds normalized degree and node-level SPD-histogram features before L2 lifting, improving CFI from $0/100$ to $10/100$. Details are in Appendix A.10.

Efficiency. We also evaluate efficiency via runtime profiling. Following common practice, we profile the top-performing lifted baselines on the same 340-pair BREC subset and RPC setting described in Appendix A.8. As shown in Tables 4 and 5, **L2GNN** has the lowest total runtime among these compared higher-order baselines under this setting.

6.2 Real-world graph classification

Datasets and protocol. We evaluate **L2GNN** on two OGB benchmarks [17], namely `ogbg-molhiv` (ROC-AUC) and `ogbg-molpcba` (128-task AP), and report mean$\pm$std over 10 runs. Complementary TU benchmark results [25] are summarized in Table 3 (full comparisons in Appendix A.11).

Models. We compare against representative *single-model* OGB leaderboard baselines without external data or pretraining, including MPNN-style baselines (GCN, GIN, GIN+VN, PNA [8], GINE+VN [4]) and higher-order methods (GSN / directional GSN [3], CIN / CIN-small [2], GIN-AK [38]). Ensembles, fingerprint-augmented, and pretrained entries are excluded for fair comparison.

Results. Table 2 reports OGB results. On `ogbg-molhiv`, **L2GNN** obtains the highest mean Test ROC-AUC in Table 2. On `ogbg-molpcba`, L2GNN obtains the second-best mean AP in Table 2 (0.2921 ± 0.0024), slightly below GIN-AK (0.2930 ± 0.0044), while improving over MPNN-style baselines such as GIN+VN, PNA, and GINE+VN. On TU (Table 3), L2GNN attains the best or second-best mean accuracy on all six evaluated datasets. These results indicate that walk-state lifting is practically competitive on real-world graph classification tasks, while BREC remains the main empirical evidence for realized expressiveness.

Table 2: OGB results on the official scaffold split. We report ROC-AUC for `ogbg-molhiv` and AP for `ogbg-molpcba`. Missing entries are unreported, best is bold, and second-best is underlined.

Model	`molhiv` (ROC-AUC)	`molpcba` (AP)
MPNN-style baselines		
GCN	0.7606 ± 0.0097	0.2020 ± 0.0024
GIN	0.7558 ± 0.0140	0.2266 ± 0.0028
GIN+VN	0.7707 ± 0.0149	0.2703 ± 0.0023
PNA	0.7905 ± 0.0132	0.2838 ± 0.0035
GINE+VN	–	0.2917 ± 0.0015
Higher-order baselines		
GSN	0.7799 ± 0.0100	–
directional GSN	0.8039 ± 0.0090	–
CIN-small	0.8055 ± 0.0104	–
CIN	$\underline{0.8094 \pm 0.0057}$	–
GIN-AK	–	$\mathbf{0.2930 \pm 0.0044}$
L2GNN (ours)	$\mathbf{0.8146 \pm 0.0048}$	$\underline{0.2921 \pm 0.0024}$

Table 3: Compact TU benchmark summary. We compare L2GNN with GIN and the strongest previously reported higher-order baseline for each dataset. Full comparisons are in Appendix A.11.

Dataset	GIN	Best prior HO	L2GNN
MUTAG	89.4 ± 5.6	CIN: $\underline{92.7 \pm 6.1}$	$\mathbf{93.1 \pm 5.0}$
PTC_MR	64.6 ± 7.0	CIN: $\mathbf{68.2 \pm 5.6}$	$\underline{66.6 \pm 6.7}$
PTC_FM	–	1-2-3 GNN: $\underline{62.8}$	$\mathbf{64.5 \pm 6.4}$
ENZYMES	–	CoSp-PPGN: $\mathbf{74.5 \pm 6.5}$	$\underline{73.3 \pm 5.6}$
NCI1	82.7 ± 1.7	CIN: $\underline{83.6 \pm 1.4}$	$\mathbf{84.9 \pm 2.3}$
NCI109	–	CIN: $\underline{84.0 \pm 1.6}$	$\mathbf{84.6 \pm 1.6}$

Scalability. `ogbg-molpcba` ($\sim$438K graphs) is a stress test for higher-order methods. With our default sparse/local configuration, L2GNN trains end-to-end on a single RTX 3090; detailed hyperparameters and the configuration are given in Appendix A.13.

6.3 Ablation analysis

Removing local closure reduces the BREC score from $297/400$ to $273/400$, whereas two-step closure yields $298/400$. Removing the structure-aware token reduces the score to $283/400$. These results support both design choices; full ablations are reported in Appendix A.10.

7 Conclusion

We proposed L2GNN, a walk-state higher-order message passing framework with candidate-indexed FWL-style bundling. It refines a symmetrized localized 3-FWL procedure and can separate certain 3-WL-indistinguishable strongly regular pairs via K_4 detection. Empirically, L2GNN achieves the highest realized-expressiveness score among the compared baselines on BREC and competitive performance on real-world graph classification tasks including the OGB and TU benchmarks.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China under Grants 62676036, 62576371, 62622615 and 62576198, and the Open Project Foundation of Key Laboratory of Computation Intelligence and Chinese Information Processing of Ministry of Education and Key Laboratory of Data Intelligence and Cognitive Computing of Shanxi Province. The authors declare no competing interests.

References

[1] V. Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On weisfeiler-leman invariance: Subgraph counts and related graph properties, 2019. URL https://arxiv.org/abs/1811.04801.

[2] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. Weisfeiler and Lehman Go Cellular: CW Networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 2625–2640, 2021.

[3] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022.

[4] Rémy Brossard, Oriel Frigo, and David Dehaene. Graph Convolutions That Can Finally Model Local Structure. *CoRR*, abs/2011.15069, 2020. URL https://arxiv.org/abs/2011.15069.

[5] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An Optimal Lower Bound on the Number of Variables for Graph Identification. *Combinatorica*, 12(4):389–410, 1992.

[6] Dexiong Chen, Till Hendrik Schulz, and Karsten Borgwardt. Learning long range dependencies on graphs via random walks, 2024. URL https://arxiv.org/abs/2406.03386.

[7] Rongqin Chen, Fan Mo, Pak Lon Ip, Shenghui Zhang, Dan Wu, Ye Li, and Leong Hou U. Connectivity-guided sparsification of 2-fwl gnns: Preserving full expressivity with improved efficiency, 2025. URL https://arxiv.org/abs/2511.12838.

[8] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velickovic. Principal Neighbourhood Aggregation for Graph Nets. *CoRR*, abs/2004.05718, 2020. URL https://arxiv.org/abs/2004.05718.

[9] Lin Du, Lu Bai, Jincheng Li, Lixin Cui, Hangyuan Du, Lichi Zhang, Yuting Chen, and Zhao Li. LGAN: An efficient high-order graph neural network via the line graph aggregation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 20914–20922, 2026. doi: 10.1609/aaai.v40i25.39232. URL https://ojs.aaai.org/index.php/AAAI/article/view/39232.

[10] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. *arXiv preprint arXiv:1912.09893*, 2019.

[11] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How Powerful Are K-hop Message Passing Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 35, pages 4776–4790, 2022.

[12] Jiarui Feng, Lecheng Kong, Hao Liu, Dacheng Tao, Fuhai Li, Muhan Zhang, and Yixin Chen. Extending the Design Space of Graph Neural Networks by Rethinking Folklore Weisfeiler-Lehman. In *Advances in Neural Information Processing Systems*, volume 36, pages 9029–9064, 2023.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *Proceedings of the International Conference on Machine Learning*, pages 1263–1272, 2017.

[14] Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

[15] Martin Grohe. The Logic of Graph Neural Networks, 2022. URL https://arxiv.org/abs/2104.14624.

[16] Martin Grohe and Martin Otto. Pebble Games and Linear Equations, 2015. URL https://arxiv.org/abs/1204.1990.

[17] 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133, 2020.

[18] Ningyuan Teresa Huang and Soledad Villar. A Short Tutorial on the Weisfeiler–Lehman Test and Its Variants. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537. IEEE, 2021.

[19] Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. Boosting the Cycle Counting Power of Graph Neural Networks with I^2-GNNs. In *International Conference on Learning Representations*, 2023.

[20] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[21] Meng Liu, Haiyang Yu, and Shuiwang Ji. Empowering GNNs via Edge-Aware Weisfeiler-Leman Algorithm. *Transactions on Machine Learning Research*, 2024.

[22] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably Powerful Graph Networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[23] Gaspard Michel, Giannis Nikolentzos, Johannes Lutzeyer, and Michalis Vazirgiannis. Path Neural Networks: Expressive and Accurate Graph Neural Networks. In *Proceedings of the International Conference on Machine Learning*, pages 24737–24755, 2023.

[24] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[25] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A Collection of Benchmark Datasets for Learning with Graphs, 2020. URL `https://arxiv.org/abs/2007.08663`.

[26] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and Leman Go Sparse: Towards Scalable Higher-Order Graph Embeddings. In *Advances in Neural Information Processing Systems*, volume 33, pages 21824–21840, 2020.

[27] Seonghyun Park, Narae Ryu, Gahee Kim, Dongyeop Woo, Se-Young Yun, and Sungsoo Ahn. Non-backtracking graph neural networks, 2024. URL `https://arxiv.org/abs/2310.07430`.

[28] Harishchandra S Ramane, Hanumappa B Walikar, Siddani Bhaskara Rao, B Devadas Acharya, Prabhakar Ramrao Hampiholi, Sudhir R Jog, and Ivan Gutman. Spectra and Energies of Iterated Line Graphs of Regular Graphs. *Applied Mathematics Letters*, 18(6):679–682, 2005.

[29] Ryoma Sato. A Survey on The Expressive Power of Graph Neural Networks. *arXiv preprint arXiv:2003.04078*, 2020.

[30] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(9):2539–2561, 2011.

[31] Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Walking out of the weisfeiler leman hierarchy: Graph learning beyond message passing. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=vgXnEyeWVY`.

[32] Yanbo Wang and Muhan Zhang. An Empirical Study of Realized GNN Expressiveness. In *Proceedings of the International Conference on Machine Learning*, 2024.

[33] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the International Conference on Machine Learning*, pages 5453–5462, 2018.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

[35] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do Transformers Really Perform Badly for Graph Representation? In *Advances in Neural Information Processing Systems*, volume 34, pages 28877–28888, 2021.

[36] Rex 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*, volume 31, 2018.

[37] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[38] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. *CoRR*, abs/2110.03753, 2021. URL `https://arxiv.org/abs/2110.03753`.

[39] Lingxiao Zhao, Louis Härtel, Neil Shah, and Leman Akoglu. A Practical, Progressively-Expressive GNN. In *Advances in Neural Information Processing Systems*, volume 35, pages 34106–34120, 2022.

A Additional material

A.1 Proof of Proposition 5.2

Proof. We prove the claim by induction on the layer index p.

Base step. At $p = 0$, the color $c_{\mathbf{v}}^{(0)}$ is the initial triplet color, which consists of the node/edge-label multiset together with the induced structure on $\mathrm{supp}(\mathbf{v})$. Under the injective initial-encoding assumption in Proposition 5.2, there exists an injective map Φ_0 such that

$$\mathbf{h}_{\mathbf{v}}^{(0)} = \Phi_0(c_{\mathbf{v}}^{(0)})$$

for every $\mathbf{v} \in \mathcal{U}(G)$.

Inductive step. Assume that for some $p \geq 0$ there exists an injective map Φ_p such that

$$\mathbf{h}_{\mathbf{v}}^{(p)} = \Phi_p(c_{\mathbf{v}}^{(p)})$$

for all $\mathbf{v} \in \mathcal{U}(G)$. Fix a triplet $\mathbf{v}$ and a candidate $s \in \mathcal{S}^\star(\mathbf{v})$. By closure, $\mathbf{v}_{(i\to s)} \in \mathcal{U}(G)$ for all $i \in \{1, 2, 3\}$. Therefore, by the induction hypothesis, the multiset

$$\{\{\mathbf{h}_{\mathbf{v}_{(1\to s)}}^{(p)}, \mathbf{h}_{\mathbf{v}_{(2\to s)}}^{(p)}, \mathbf{h}_{\mathbf{v}_{(3\to s)}}^{(p)}\}\}$$

is an injective image of

$$\{\{c_{\mathbf{v}_{(1\to s)}}^{(p)}, c_{\mathbf{v}_{(2\to s)}}^{(p)}, c_{\mathbf{v}_{(3\to s)}}^{(p)}\}\} = \sigma_{\mathbf{v},s}^{(p+1)}.$$

Since the bundle combiner is an injective multiset function, there exists an injective map Ψ_p such that

$$\mathbf{g}_{\mathbf{v},s}^{(p)} = \Psi_p(\sigma_{\mathbf{v},s}^{(p+1)}).$$

Aggregating over candidates with an injective multiset function gives an injective representation of

$$\{\{\sigma_{\mathbf{v},s}^{(p+1)} \mid s \in \mathcal{S}^\star(\mathbf{v})\}\}.$$

Finally, the update function is injective and also receives the previous state $\mathbf{h}_{\mathbf{v}}^{(p)}$, which is an injective image of $c_{\mathbf{v}}^{(p)}$. Hence there exists an injective map Φ_{p+1} such that

$$\mathbf{h}_{\mathbf{v}}^{(p+1)} = \Phi_{p+1}(c_{\mathbf{v}}^{(p+1)}).$$

This completes the induction. □

A.2 Proof of Proposition 5.3

Proof. For item 1, set $\mathcal{U}(G) = V^3$ and $\mathcal{S}(\mathbf{v}) = V$. For every triplet $\mathbf{v}$ and every replacement node $s \in V$, an ordered, position-sensitive combiner receives

$$\big(c_{\mathbf{v}_{(1\to s)}}^{(p)}, c_{\mathbf{v}_{(2\to s)}}^{(p)}, c_{\mathbf{v}_{(3\to s)}}^{(p)}\big),$$

which is exactly the replacement vector used by global 3-FWL. The outer multiset aggregation over $s \in V$ is also exactly the 3-FWL update. Thus an injective ordered global L2GNN layer simulates one round of 3-FWL, and induction over the number of layers proves simulation for all rounds. Since 3-FWL is as expressive as 4-WL for $k \geq 2$, and since the WL hierarchy is strict, this ordered global variant strictly separates some pairs that 3-WL cannot.

For item 2, the default model differs from this ordered global construction in two ways. The candidate domain is reduced from V to $\mathcal{S}^\star(\mathbf{v})$, and the position-sensitive vector is replaced by an unordered multiset. Hence it is an operational localized and symmetrized restriction of global 3-FWL.

For item 3, if $\mathcal{S}^\star(\mathbf{v}) = \emptyset$, then there is no candidate-indexed replacement bundle. The implemented update sets $\mathbf{m}_{\mathbf{v}}^{(p)} = \mathbf{0}$, so the next state is only an MLP/residual transform of the current triplet state. No new graph witness is queried.

For item 4, if all candidates lie inside $\mathrm{supp}(\mathbf{v})$, every replacement remains within the nodes already present in the base triplet. Therefore the layer can only recombine information already encoded in the base triplet and cannot inspect an external witness node. □

A.3 Proof of Proposition 5.4

This proof uses only the K_4-detection subcase of Proposition 5.4; it does not require recovering the full labeled four-node adjacency pattern.

Proof. Let $H = G[\{v_1, v_2, v_3, s\}]$. The four induced three-node subgraphs obtained by deleting one vertex from H are

$$G[\{v_1, v_2, v_3\}], \quad G[\{s, v_2, v_3\}], \quad G[\{v_1, s, v_3\}], \quad G[\{v_1, v_2, s\}].$$

The first one is the base triplet structure, and the last three are exactly the structural replacement bundle $\mathcal{B}_G(\mathbf{v}, s)$. Therefore, the base triplet together with $B_G(\mathbf{v}, s)$ determines all four induced three-node subgraphs obtained by deleting one vertex from the four-node extension.

For the K_4 subcase, assume $G[\{v_1, v_2, v_3\}] \cong K_3$. Then $G[\{v_1, v_2, v_3, s\}] \cong K_4$ if and only if all three replacement triplets in $\mathcal{B}_G(\mathbf{v}, s)$ are also triangles. This condition is determined by the induced three-node isomorphism types encoded in the bundle. □

A.4 Proof of Proposition 5.5

Proof. Let $\mathbf{v} = (v_1, v_2, v_3)$ be a triangular triplet, i.e., $\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\} \in E(G)$, and let $s \in \mathcal{S}^\star(\mathbf{v}) \setminus \mathrm{supp}(\mathbf{v})$ be an external candidate. By complete-bundle filtering, all three replacement triplets

$$(s, v_2, v_3), \qquad (v_1, s, v_3), \qquad (v_1, v_2, s)$$

are included in the candidate bundle.

Since the initial encoder is injective on induced structures, the bundle distinguishes whether each replacement triplet is a triangle. The triplet (s, v_2, v_3) is a triangle iff $\{s, v_2\}, \{s, v_3\} \in E(G)$; similarly, (v_1, s, v_3) is a triangle iff $\{s, v_1\}, \{s, v_3\} \in E(G)$, and (v_1, v_2, s) is a triangle iff $\{s, v_1\}, \{s, v_2\} \in E(G)$. Therefore, all three replacement triplets are triangles iff s is adjacent to all of v_1, v_2, v_3. Together with the base triangle on $\{v_1, v_2, v_3\}$, this is equivalent to $G[\{v_1, v_2, v_3, s\}] \cong K_4$.

Thus each candidate-indexed K_4 extension produces a detectable bundle signature. Under canonicalized triplets, each K_4 contributes a fixed positive multiplicity of such signatures; under ordered triplets, the multiplicity is different but still fixed. By injectivity of the candidate aggregation, state update, and graph-level readout, graphs with different K_4 counts obtain different graph-level representations.

The 4×4 Rook's graph contains K_4 subgraphs induced by its four rows and four columns. The Shrikhande graph has clique number 3 and therefore contains no K_4. Thus default L2GNN separates the pair. Since this pair is not separated by the 3-WL convention used in this paper, default L2GNN is not upper bounded by 3-WL. □

A.5 Illustration of sparsification, localization, and FWL-style bundling

Remark A.1 (On the role of sparsification, localization, and FWL-style bundling). Proposition 5.3 gives the expressive upper bound of an ordered global instantiation (dense V^3 with $\mathcal{S}(\mathbf{v}) = V$). In practice, however, operating on the full triplet space is not only computationally prohibitive, but also not necessarily useful. Figure 3 (third column) shows that a designed sparsified and localized instantiation same as our default choice, i.e., $w \in \mathcal{N}(v_1) \cup \mathcal{N}(v_2) \cup \mathcal{N}(v_3)$ for a base triplet (v_1, v_2, v_3), yields a markedly different candidate set (blue) between the Shrikhande and 4×4 Rook's graphs, thereby breaking the regular symmetry and enabling separation without the dense V^3 domain.

In addition, we further adopt 3-FWL-style bundling instead of 3-WL-style aggregation because our designed bundling captures induced 4-node patterns from triplet states. In Fig. 3, 3-WL aggregates single-position replacements position-wise and essentially keeps only counts, thus losing whether the replacements across positions come from the same node w (cf. "counts only; identity lost across positions"). In contrast, 3-FWL couples the three replacements induced by the same w into a candidate-indexed bundle, equivalently reasoning about the induced 4-node configuration on $\{1, 2, 3, w\}$, which can be crucial for complex 4-node structures (e.g., 4-clique-related motifs).

A.6 Complexity of sparsified and localized L2GNN

Let $\mathcal{T}_2(G)$ be the lifted triplet set used by L2GNN and $T = |\mathcal{T}_2(G)|$. For each triplet $\mathbf{v}$, let $\mathcal{S}^\star(\mathbf{v})$ be the filtered candidate set and define $K = \max_{\mathbf{v}} |\mathcal{S}^\star(\mathbf{v})|$.

Lifted domain size. The seed triplets are non-backtracking 2-hop walks, so before canonicalization

$$T_{\text{seed}} = O\left(\sum_{v \in V} d_v(d_v - 1)\right) = O\left(\sum_{v \in V} d_v^2\right).$$

The final default domain is not only the seed set: it also includes one-step local-closure triplets induced by selected replacement candidates. Therefore, before deduplication,

$$T \le \min\left\{ O(n^3),\ T_{\text{seed}} + 3 \sum_{\mathbf{v} \in \mathcal{T}_2^{\text{seed}}(G)} |\mathcal{S}(\mathbf{v})| \right\} \le \min\left\{ O(n^3),\ O\left(\Delta \sum_{v \in V} d_v^2\right) \right\},$$

where $\Delta = \max_{v \in V} d_v$. Canonicalization and deduplication can only reduce this count in the undirected setting. Thus the seed domain scales as $O(\sum_v d_v^2)$, while the one-step locally closed domain has an additional worst-case Δ factor. In bounded-degree or moderate-degree graphs, this remains close to the seed complexity.

Per-layer time/space. Let

$$M = \sum_{\mathbf{v} \in \mathcal{T}_2(G)} |\mathcal{S}^\star(\mathbf{v})|$$

be the number of retained candidate bundles. Each retained candidate bundle contains three single-position replacements, so the number of lifted replacement edges is at most $3M \le 3KT$, where $K = \max_{\mathbf{v}} |\mathcal{S}^\star(\mathbf{v})|$. Therefore, each layer costs $O(M)$ time and $O(T + M)$ memory, or equivalently $O(KT)$ time and $O(T + KT)$ memory in the worst case, ignoring feature dimensions. In the default candidate scope, $K = O(\Delta)$.

Comparison to 3-WL/3-FWL. Global 3-WL/3-FWL maintains colors on all V^3 triplets ($\Theta(n^3)$ states) and aggregates over $\Theta(n)$ replacements per triplet, yielding $\Theta(n^4)$ time per iteration. In contrast, default L2GNN operates on the retained locally closed domain $\mathcal{T}_2(G)$ and costs $O(KT)$ per layer. Under the one-step closure bound above, this is $O\big(K \min\{n^3, \Delta \sum_v d_v^2\}\big)$ in the worst case, and is substantially smaller on sparse or moderate-degree graphs.

A.7 Additional experimental details

Hardware Specification: All experiments were performed on a machine with a single RTX 3090 GPU (24GB VRAM). CPU is Intel Core i5-12490F, and RAM is 128GB. Storage is 4TB NVMe SSD.
Reproducibility: To facilitate the reproduction of our evaluation results, we provide the complete source code and instructions (README) in the supplementary material submitted to OpenReview.

For models that do not report results on BREC, we run their official code with recommended hyperparameters and report the best results across multiple runs. For some models, we also report results from the original papers when available.

A.8 Efficiency and runtime experiments

To evaluate efficiency in a commonly used BREC cost-profiling setting, we profile runtime on a BREC subset. Specifically, we time the top-3 models on a 340-pair subset that excludes 4–Vertex Condition pairs and keeps 60 CFI pairs, using `brec_v3_no4v_60cfi.npy` sourced from the public BREC repository in place of the full `brec_v3.npy` to reduce data-loading overhead. In BREC's taxonomy, the Regular category comprises simple regular, strongly regular, 4–Vertex Condition, and Distance Regular graphs, and we use *Regular-core* to denote the combined subset of simple and strongly regular graphs (100 pairs in total).

Table 4: **Runtime on the evaluated BREC subset (minutes).**

Method	Preproc. (min)	Eval. (min)	Total (min)
L2GNN	**80.78**	**40.56**	**121.34**
I^2-GNN	90.99	61.92	152.91
N^2-GNN	164.34	383.44	547.78

Table 5: **Evaluation time breakdown by BREC split (minutes).**

Method	Basic (60)	Regular-core (100)	Extension (100)	Distance Regular (20)	CFI (60)	Total (340)
L2GNN	0.85	16.27	1.60	**9.39**	**12.44**	**40.56**
I^2-GNN	**0.24**	**3.89**	**0.27**	17.91	39.61	61.92
N^2-GNN	0.38	16.94	0.68	62.18	303.25	383.44

We also record the on-disk footprint of cached preprocessing outputs (measured on the output directory, uncompressed): I^2-GNN/L2GNN/N^2-GNN use 51.2/90.1/112.0 GB on this 340-pair subset, respectively. We cache intermediate indexing tensors (e.g., candidate tuple lists and sparse adjacency) to amortize preprocessing across the repeated runs required by the BREC/RPC protocol (pairwise training and multiple seeds/permutations). This disk cache is an optional acceleration for reproducibility; precomputing and caching tuple-index structures is commonly used in higher-order (k-tuple) message-passing/k-WL-based implementations to amortize repeated access to the same indexing patterns. The same structures can also be generated on-the-fly at the cost of additional compute. Our candidate restriction and complete-bundle filtering are designed to keep the lifted domain sparse and well-formed, mitigating the combinatorial blow-up of tuple states while preserving the intended localized message-passing semantics. Empirically, this yields a favorable expressiveness–efficiency trade-off on BREC: among the compared lifted baselines, L2GNN achieves the strongest overall distinguishing accuracy while using a mid-range cache footprint on the same subset.

A.9 Lifted-graph preprocessing

This section summarizes the preprocessing steps used to construct lifted states (triplets) and lifted edges in our experiments.

Algorithm 1 Lifted-graph preprocessing used in our experiments (high-level).

Require: Graph data with `edge_index` (and optional `x`, `edge_attr`); preprocessing knobs as described in Appendix A.9.
Ensure: Lifted triplets $\mathcal{T}$, lifted edges $\mathcal{E}_2$, and lifted node features.
Build structural edge set E_{struct} (optionally make it bidirectional).
Enumerate non-backtracking 2-hop walks (v_1, v_2, v_3) with $v_1 \neq v_3$ to form initial triplets.
Deduplicate triplets; if `strict_dedup=True`, sort each triplet then dedup.
if `expand_triplets` **then**
    for $s = 1, \ldots,$ `expand_steps` **do**
        Expand triplets by local replacements using candidate sets controlled by `shared_cand_mode`.
        Stop if reaching `max_triplets_per_graph`.
    end for
end if
Build lifted node features: use invariant sum/max/mean pooling of triplet node features when `node_feature_mode=invariant`, otherwise use ordered coordinate concatenation; concatenate the resulting node-feature block with the structure-aware token.
Build lifted edges with type labels $0/1/2$ for the three replacement positions.
if `require_complete_w` **then**
    Keep (dst, k) only if all three types exist.
end if

Preprocessing knobs used in our experiments. Unless stated otherwise, we use `make_undirected=True`, `struct_from_raw=False`, `strict_dedup=True`, `canonicalize_endpoints=False`; `expand_triplets=True`, `expand_steps=1`,

`shared_cand_mode=`$\mathtt{v}_1\mathtt{v}_2\mathtt{v}_3$, `require_complete_w=True`; `node_feature_mode=auto`; and `max_k_per_pos=0`, `max_triplets_per_graph=5e10` (no effective cap in our experiments: `0` disables the former limit and `5e10` is a large sentinel for the latter).

With `strict_dedup=True`, `node_feature_mode=auto` resolves to the invariant mode, which represents triplet node features by the concatenation of coordinate-wise sum, max, and mean over the three nodes. With `strict_dedup=False`, `auto` uses ordered coordinate concatenation. This matches the distinction between the canonicalized default model and the practical ordered variant.

L2 input features in the structure-only setting. For structure-only graphs with no node or edge attributes, we use constant node features ($x_v = \mathbf{1}$), together with the induced-subgraph structural token described in Section 4.

A.10 Ablation studies

We further conduct controlled ablations to isolate the effects of shared candidate set scope, local closure, ordered triplet states, the structure-aware state encoding and some additional components.

Shared candidate-set scope (`shared_cand_mode`). In Algorithm 1, we build the lifted graph by expanding each triplet node to a local candidate set. The hyperparameter `shared_cand_mode` controls how the shared candidate set is formed: $\mathtt{v}_1\mathtt{v}_2\mathtt{v}_3$ uses $N(v_1) \cup N(v_2) \cup N(v_3)$, $\mathtt{v}_1\mathtt{v}_2$ uses $N(v_1) \cup N(v_2)$, and $\mathtt{v}_2$ uses $N(v_2)$ only. We ablate this design choice on the BREC evaluation subset (Basic/Regular-core/Extension, 260 pairs total) under the same evaluation protocol and hyperparameter settings, since preprocessing the full dataset is computationally demanding, and this subset already provides a clear comparison across splits.

Table 6: Ablation on the scope of the shared local candidate set (`shared_cand_mode`) on the BREC evaluation subset.

`mode`	Basic (60)	Regular-core (100)	Extension (100)	Total (260)
$\mathtt{v}_1\mathtt{v}_2\mathtt{v}_3$	**60/60 (100%)**	**100/100 (100%)**	**100/100 (100%)**	**260/260 (100%)**
$\mathtt{v}_1\mathtt{v}_2$	8/60 (13%)	25/100 (25%)	6/100 (6%)	39/260 (15%)
$\mathtt{v}_2$	0/60 (0%)	5/100 (5%)	0/100 (0%)	5/260 (2%)

Effect of local closure depth. The default preprocessing performs one step of local closure expansion so that replacement-style aggregation can form complete candidate-indexed bundles. To test whether additional closure improves realized expressiveness, we compare no closure, the default one-step closure, and two-step closure on BREC. Table 7 shows that removing closure substantially hurts performance, reducing the total score from $297/400$ to $273/400$. Increasing the closure depth from one to two steps brings only a marginal gain, from $297/400$ to $298/400$, **with the improvement confined to the CFI split**. This suggests that one-step local closure already captures most of the useful local replacement structure under the current budget, while further closure gives diminishing returns and increases preprocessing cost.

Table 7: Ablation on the number of local closure expansion steps on BREC. The default L2GNN uses one-step closure. Each entry reports the number of distinguished pairs.

Variant	Basic (60)	Regular (140)	Extension (100)	CFI (100)	Total (400)
No closure	60	120	93	0	273
1-step closure (default)	**60**	**137**	**100**	0	297
2-step closure	**60**	**137**	**100**	**1**	**298**

Practical ordered variant. We also evaluate a practical ordered variant by turning off the default symmetrization/deduplication in the implementation, while keeping the same hyperparameter setting. This variant should not be confused with the dense ordered global model in Proposition 5.3, which uses the full triplet domain and global replacement candidates and is not practically feasible under our compute budget. As shown in Table 8, the practical ordered variant improves the total BREC score only slightly, from $297/400$ to $299/400$. The gain is confined to the Regular split, increasing

Table 10: CFI diagnostic comparison on BREC. The SPD-profile variant is an optional diagnostic extension and is not used as the default model in the main comparison.

Method	Extra signal / scope	CFI
L2GNN (default)	local walk-state	0/100
L2GNN + SPD-profile	local walk-state + node SPD histogram + degree	10/100

from $137/140$ to $139/140$, while Basic, Extension, and CFI remain unchanged. The ordered variant is substantially more expensive because it keeps multiple ordered states that are merged in the default symmetrized version; for example, it keeps two states for an open 3-node path and six states for a triangle, instead of merging each into one canonical state. Therefore, the default symmetrized localized model remains the better expressiveness–efficiency tradeoff.

Table 8: Ablation of the default symmetrized/canonicalized triplet representation versus a practical ordered variant on BREC. The practical ordered variant disables the default symmetrization/deduplication, but is still sparse and local.

Variant	Basic (60)	Regular (140)	Extension (100)	CFI (100)	Total (400)
Symmetrized default	**60**	137	**100**	**0**	297
Practical ordered variant	**60**	**139**	**100**	**0**	**299**

Effect of structure-aware state encoding. We ablate the structure-aware state encoding by removing only the induced-subgraph structural token from the lifted triplet initialization, while keeping the same node features, lifted graph construction, aggregation, and hyperparameter setting. Since BREC uses constant node features, this node-feature-only variant isolates the contribution of the induced-subgraph structural encoding. Table 9 shows that the score drops from $297/400$ to $283/400$, with the largest drop on the Regular split. This indicates that the structure-aware encoding contributes materially to the realized expressiveness of L2GNN, especially on regular graph pairs, but it is not the sole source of distinguishing power.

Table 9: Ablation of the structure-aware state encoding on BREC. The node-feature-only variant removes the induced-subgraph structural encoding while keeping the same hyperparameter setting. Since BREC node features are constant, this isolates the contribution of the structure-aware token.

Variant	Basic (60)	Regular (140)	Extension (100)	CFI (100)	Total (400)
Node-feature-only	56	128	99	**0**	283
Structure-aware default	**60**	**137**	**100**	**0**	**297**

Optional SPD-profile extension on CFI. The default L2GNN is intentionally sparse and local, and therefore does not use graph-wide positional or distance encodings. To test whether additional global structural information can partially mitigate the CFI limitation, we evaluate a diagnostic SPD-profile variant. This variant augments the original node features before L2 lifting with a normalized degree scalar and a node-level SPD histogram with cutoff 8, while keeping the same L2 lifting and message-passing pipeline. Table 10 reports the CFI ablation.

Additional GNN baselines with positional/structural encodings. We further compare against standard MPNN/Transformer baselines augmented with graph-wide positional or structural encodings. Specifically, we evaluate GCN, GIN, and Transformer backbones with RRWP8 or LapPE8 under the same BREC RPC protocol, matched backbone settings, and matched training budget. As shown in Table 11, these PE-augmented standard baselines are substantially lighter than higher-order lifted models, but they lose a large amount of realized expressiveness on BREC. The strongest among them is GIN+RRWP8 with $208/400$, far below L2GNN's $297/400$. This supports the scope of our efficiency claim: L2GNN is not intended to be as cheap as standard 1-WL backbones, but to provide a favorable expressiveness–efficiency tradeoff among practical higher-order methods.

Table 11: Additional standard MPNN/Transformer baselines with graph-wide positional or structural encodings on BREC. Each entry reports the number of distinguished pairs over the total number of graph pairs in the corresponding split. All methods are evaluated under the same BREC RPC protocol with matched backbone settings and training budget. Best results are in bold.

Model	Basic	Regular	Extension	CFI	Total
GCN + RRWP8	60/60	48/140	99/100	0/100	207/400
GIN + RRWP8	60/60	49/140	99/100	0/100	208/400
Transformer + RRWP8	60/60	48/140	99/100	0/100	207/400
GCN + LapPE8	44/60	37/140	79/100	0/100	160/400
GIN + LapPE8	52/60	40/140	82/100	0/100	174/400
Transformer + LapPE8	45/60	17/140	66/100	0/100	128/400
L2GNN (ours)	**60/60**	**137/140**	**100/100**	**0/100**	**297/400**

A.11 Additional TU benchmark results

Table 12: Graph classification accuracies (%). We report the mean accuracy and, when available, the standard deviation. Best performances are in bold and second-best are underlined. `N/A` indicates results not reported for the corresponding dataset in the original paper.

Datasets		MUTAG	PTC_MR	PTC_FM	ENZYMES	NCI1	NCI109
	# graphs	188	344	349	600	4110	4127
	# classes	2	2	2	6	2	2
	Avg. # nodes	18	14	14	33	30	30
Non-GNN	3-WL	83.2	62.5	61.5	42.3	83.1	N/A
1-WL-based	GIN (2019)	89.4±5.6	64.6±7.0	N/A	N/A	82.7±1.7	N/A
Beyond-1-WL	1-2-3 GNN (2019)	86.1	60.9	<u>62.8</u>	N/A	76.2	N/A
	PPGN (2019)	90.6±8.7	66.2±6.5	N/A	N/A	83.2±1.1	81.8±1.9
	δ-3-LWL$^+$ (2020)	N/A	N/A	61.3±2.0	56.8±1.2	82.7±0.5	81.9±0.4
	CIN (2021)	<u>92.7±6.1</u>	**68.2±5.6**	N/A	N/A	<u>83.6±1.4</u>	<u>84.0±1.6</u>
	PathNN (SP$^+$) [23] (2023)	N/A	N/A	N/A	70.4±3.1	82.3±1.9	N/A
	NC-GNN [21] (2024)	90.6±5.6	64.4±5.6	N/A	N/A	82.5±1.9	N/A
	CoSp-PPGN (2025)	N/A	N/A	N/A	**74.5±6.5**	82.9±1.9	82.9±1.2
	L2GNN	**93.1**±5.0	<u>66.6±6.7</u>	**64.5±6.4**	<u>73.3±5.6</u>	**84.9±2.3**	**84.6±1.6**

Datasets and protocol. For completeness with the broader graph classification literature, we additionally evaluate L2GNN on six widely used TU benchmarks [25]: MUTAG, PTC_MR, PTC_FM, ENZYMES, NCI1, NCI109. Following the standard protocol in [34], we adopt 10-fold cross-validation and report mean±std accuracy across folds. We note that, as observed in [10], TU benchmarks are known to suffer from high variance and limited statistical resolution; we therefore present these results as *complementary downstream evidence* rather than the primary support for L2GNN's empirical claims, which rest on BREC (Section 6.1) and OGB (Section 6.2).

Results. Table 12 summarizes the TU classification results. L2GNN achieves the best or second-best mean accuracy on all six datasets where it is evaluated, while remaining within one standard deviation of the strongest baselines on the smaller datasets (MUTAG, PTC, ENZYMES) where TU variance is largest. Consistent with prior observations on TU [10, 34], stronger worst-case expressiveness beyond 1-WL does not translate into a monotonic improvement in TU accuracy, and the 1-WL baseline GIN remains highly competitive on several datasets. We therefore caution against drawing strong conclusions from TU alone; the more rigorous evidence for L2GNN's practical utility is given by BREC and OGB.

A.12 BREC experimental setup

Training and evaluation hyperparameters. We use `NUM_RELABEL`=32 relabelings per graph and `SAMPLE_NUM`=400 for the reliability offset. For each gid we train for at most `EPOCH`=20 epochs with batch size `BS`=4, Adam learning rate `LR`=10^{-3}, and weight decay `WD`=10^{-4}. The embedding dimension is `OUTPUT_DIM`=16. We apply early stopping when the average base loss is below

LOSS_THRESHOLD=0.2. For the Hotelling-T^2 decision we use threshold $\tau = \texttt{THRESHOLD} = 72.34$. We set MARGIN=0, variance regularization weight var_w=0.1, and GRAD_ACCUM_STEPS=2.

Default model configuration. Unless stated otherwise, we use num_layers=6, hidden_dim=16, dropout=0.0, pooling=add, and layer_type=fwl.

Part split. We follow the official BREC split by gid: Basic (0–60), Regular-core (60–160), Extension (160–260), CFI (260–360), 4–Vertex Condition (360–380), and Distance Regular (380–400). In BREC's taxonomy, *Regular (overall, 140)* = Regular-core + 4–Vertex Condition + Distance Regular, while *Regular-core (100)* refers to gid 60–160 only.

A.13 OGB experimental setup

Protocol. We follow the official OGB protocol for both ogbg-molhiv and ogbg-molpcba: the standard scaffold split, the official Evaluator (ROC-AUC for molhiv; AP for molpcba), and 10 runs with different random seeds. For each run, we select the best-validation epoch and report the corresponding test metric; the final entries in Table 2 are mean±std across the 10 seeds.

Fixed L2 lifting configuration. The L2 lifting hyperparameters are kept *fixed* across the entire grid so that the lifted-graph cache is built once per dataset and reused. Concretely, we use the default setting described in Section 4: 2-hop non-backtracking walk seeds with canonicalization (strict_dedup=True), one-step local closure expansion (expand_steps=1), shared local candidate set $\mathcal{S}(\mathbf{v})=\mathcal{N}(v_1)\cup\mathcal{N}(v_2)\cup\mathcal{N}(v_3)$ (shared_cand_mode=(v_1, v_2, v_3)), complete-bundle filtering (require_complete_w=True), undirected lifting (make_undirected=True), and the structure-aware token enabled (use_struct=True). We use the raw OGB atom/bond features as node/edge inputs, with node_feature_mode=auto resolving to the invariant node-feature summary because strict_dedup=True. No extra positional encoding, all-pairs distance, or pretraining is used.

Hyperparameter grid. On top of the fixed L2 cache, we perform a dataset-specific grid search over model and optimizer hyperparameters. The grid is defined once per dataset and covers the main axes that affect graph-level prediction. Adam is used throughout, with up to 100 epochs, early stopping based on validation metric (patience=30, min_delta=1e-5), and gradient clipping disabled. Table 13 summarizes the search space; all combinations in the grid are enumerated and ranked by validation metric.

Table 13: Hyperparameter grid for the OGB experiments. The L2 lifting configuration is fixed across the grid (see text). The reported numbers in Table 2 use the configuration that attains the best validation metric within this grid.

Hyperparameter	ogbg-molhiv	ogbg-molpcba
num_layers	$\{4, 5, 6, 7, 8, 9\}$	$\{4, 5, 6, 7, 8, 9\}$
hidden_dim	$\{64, 128\}$	$\{64, 128\}$
dropout	$\{0.0, 0.1, 0.2\}$	$\{0.0, 0.1\}$
pooling	{add, addmax}	{add, addmax}
layer_type	{fwl, wl, l2}	{fwl}
lr	$\{1\times10^{-3}, 5\times10^{-4}, 3\times10^{-4}\}$	$\{1\times10^{-3}, 5\times10^{-4}, 3\times10^{-4}\}$
weight_decay	$\{0,\ 1\times10^{-5}\}$	$\{0,\ 1\times10^{-5}\}$
batch_size	$\{32, 64\}$	$\{64\}$
Total configurations	2,592	288

The narrower grid for ogbg-molpcba reflects its substantially larger scale (roughly 437,929 graphs vs. 41,127 for molhiv); we therefore restrict layer_type to the default fwl variant (the symmetrized localized 3-FWL aggregation analyzed in Section 5) and fix batch_size to 64. The remaining axes are kept identical across datasets so the grid is directly comparable.

Best configurations. Table 14 reports the best-validation configurations selected from the grid in Table 13. These configurations are used to produce the OGB results in Table 2. For exact reproduction of the reported numbers, one can use Table 14; for independent model selection or transfer to new

datasets, we recommend searching over the grid in Table 13. Since the grid is finite and some selected values lie on the boundary of the searched range, the reported configurations should be interpreted as the best settings found within this search space rather than as globally optimal hyperparameters.

Table 14: Best-validation configurations selected from the grid in Table 13.

Dataset	layers	hidden	dropout	pooling	layer_type	lr	wd	bs
ogbg-molhiv	6	128	0.1	addmax	fwl	5×10^{-4}	1×10^{-5}	32
ogbg-molpcba	7	128	0.1	add	fwl	5×10^{-4}	1×10^{-5}	64

A.14 TU experimental setup

We follow 10-fold stratified cross-validation: for each fold, we select the checkpoint/epoch by validation accuracy on a split of the training data and report the corresponding test-fold accuracy; results are summarized as mean$\pm$std over folds.

Table 15: Selected hyperparameters for TU benchmarks.

Dataset	#Layers	Hidden	LR	Dropout	Weight decay
MUTAG	7	32	1e-3	0.1	0
PTC_MR	6	64	1e-3	0.0	1e-5
PTC_FM	3	32	1e-3	0.0	0
ENZYMES	3	128	1e-3	0.1	1e-5
NCI1	3	128	5e-4	0.2	0
NCI109	5	32	1e-3	0.1	1e-5

A.15 The BREC Benchmark and the CFI Split

We summarize key properties of the BREC benchmark that are relevant for interpreting our experimental results.

Composition. BREC contains 400 pairs of 1-WL-indistinguishable graphs partitioned into four splits of increasing structural difficulty. The Basic split has 60 pairs of small 1-WL-indistinguishable graphs. The Regular split has 140 pairs of regular graphs spanning simple regular, strongly regular, 4-vertex condition, and distance regular subfamilies. The Extension split has 100 pairs probing the gap between 1-WL and 3-WL. The CFI split has 100 Cai-Fürer-Immerman pairs spanning up to 4-WL-indistinguishable. Graph sizes range from 10 to 198 nodes. BREC is significantly harder than EXP, CSL, and SR25 in difficulty, granularity, and scale.

Evaluation protocol. BREC departs from classification. Each pair is tested individually via the Reliable Paired Comparison protocol. For each pair, q=32 relabelings of each graph are generated, a Siamese GNN is trained with contrastive loss, and a Hotelling T^2 statistic at confidence α=0.95 decides distinguishability. A reliability check on isomorphic copies is performed in parallel. A pair counts as distinguished only if the external T^2 exceeds a threshold while the internal T^2 remains below it. This makes BREC accuracy a conservative measure of realized expressiveness.

Why CFI is hard for local methods. The BREC study reports that local WL-style methods perform poorly on CFI due to insufficient receptive fields, while methods with a global k-WL-style receptive field benefit from global processing. CFI graphs in BREC are larger than the other splits, with up to 198 nodes, and the indistinguishability level reaches 4-WL. Distinguishing CFI pairs requires a receptive field that scales with the CFI backbone diameter. Methods restricted to local neighborhoods systematically underperform on CFI relative to their performance on Basic, Regular, and Extension. Tables 1 and 11 show that this pattern holds for many practical sparse/local methods and PE-augmented standard backbones. For example, GSN, δ-k-LGNN, KC-SetGNN, CIN, and the PE-augmented standard backbones score $0/100$ or close to it on CFI.

A.16 Relationship to related frameworks

Iterated line graphs. The walk-state graph in our framework is not the line graph of the line graph. The line graph $L(G)$ has nodes indexed by edges of G and edges between adjacent edges. Iterating gives $L^2(G)$, whose nodes are pairs of incident edges, and whose size grows rapidly with degree. For r-regular graphs with $r \geq 3$, the size of $L^k(G)$ grows super-polynomially in k. By contrast, our lifted state space is indexed by vertex tuples along ℓ-hop walks. For $\ell=2$, the seed domain has size $O(\sum_v d_v^2)$ before canonicalization. The default one-step locally closed domain is bounded as in Appendix A.6, namely by $\min\{O(n^3), O(\Delta \sum_v d_v^2)\}$ before deduplication. Message passing is defined by single-position vertex replacement rather than by edge adjacency. Our construction can be viewed as an alternative path beyond edge-level representation that avoids the combinatorial blow-up of iterated line graphs while still capturing $\ell+1$-node induced configurations.

Walk-based and random-walk-based GNNs. A separate line of work uses walks or paths as input features, sampled sequences, or aggregation objects rather than as lifted states. CRaWl [31] extracts subgraph features along random walks and processes them with 1D convolutions. NeuralWalker [6] treats random walks as sequences and applies sequence models to capture long-range dependencies. PathNN [23] aggregates paths emanating from nodes. A related non-backtracking model, NBA-GNN [27], updates directed edge messages without immediately backtracking to reduce redundant message flows. These methods differ from our framework in three structural ways. First, our framework uses walks to define the lifted state space on which message passing happens, while these methods use walks or paths as sampled sequences, features, or aggregation objects. Second, the L2GNN update is a localized 3-FWL-style single-position replacement aligned with the WL hierarchy, whereas walk/path-based methods do not directly implement a WL replacement refinement. Third, the inductive bias of L2GNN is local higher-order induced four-node configurations via candidate-indexed bundles, while these methods focus on walk/path sequences, long-range dependencies, or non-backtracking message flows.

Local k-WL-based GNNs. Local k-WL methods such as δ-k-LGNN and KC-SetGNN restrict aggregation to local neighborhoods. They refine generic k-tuples or bounded-size node sets over the full V^k or $\binom{V}{\leq k}$ space. Our framework differs in three ways. First, the state space is walk-induced and sparse, rather than the full V^k. The lifted domain is structure-aware by construction. Second, our framework uses FWL-style candidate-indexed bundle aggregation, where the $\ell+1$ single-position replacements induced by the same w are coupled, whereas δ-k-LGNN uses k-WL-style position-wise aggregation that loses the across-position identity of w. Third, on BREC, the local k-WL family scores 216 (δ-k-LGNN) and 211 (KC-SetGNN), while L2GNN scores 297. The Regular split shows the largest gap, with δ-k-LGNN at $50/140$ versus L2GNN at $137/140$. We therefore view our framework as a structurally distinct point in the design space rather than a small variant of generic local k-WL.

A.17 Additional discussion

Sparsification and localization. Our results suggest that the choice of sparsification and localization is crucial. Beyond reducing the prohibitive cost of global k-WL-style simulations, a well-designed local candidate set can expose informative local witnesses and help break regular symmetries through candidate-indexed bundles. At the same time, the ablations in Appendix A.10 show that not all locality choices are equally effective: narrowing the shared candidate set substantially hurts performance, while expanding closure beyond one step brings only marginal improvement under the current budget.

FWL-style bundling. The candidate-indexed FWL-style bundling is another important component. Compared with same-order WL-style aggregation that processes single-position replacements separately, bundling couples the replacements induced by the same candidate node. This allows the model to reason about induced configurations involving the base triplet and the replacement candidate, which can be important for distinguishing complex local structures.

A.18 Limitations and future work

Scope of the default model and SPD/PE extensions. The goal of the default L2GNN is to study how far a graph-native, higher-order, k-WL/FWL-style lifted message-passing model can go under a sparse and localized replacement scheme, rather than to maximize performance by adding graph-wide positional or distance features. Such features can be useful: our diagnostic SPD-profile variant, even with a cutoff of 8, improves the CFI score from $0/100$ to $10/100$, suggesting that additional distance-profile features can partially compensate for the limited receptive field of sparse/local higher-order models. However, they also introduce a different source of expressiveness and additional preprocessing cost. For example, computing full all-pairs shortest-path distances on an unweighted graph by running BFS from every node costs $O(n(n+m))$ time, and storing the full distance matrix costs $\Theta(n^2)$ memory per graph; this cost accumulates over large graph collections. Cutoff or histogram variants can reduce the stored feature size, but they still require additional distance-based preprocessing and move the evaluation away from the SPD/PE-free lifted-state design. We therefore report L2GNN+SPD-profile only as a diagnostic extension, and keep the SPD/PE-free L2GNN as the default model because it better reflects the intended expressiveness–efficiency trade-off of our framework.

Future work. A natural future direction is to make the locality and sparsification strategy adaptive rather than manually specified. For example, one could learn candidate-set selection, closure depth, or replacement scopes under an explicit compute budget. This may further improve the expressiveness–efficiency tradeoff and reduce the need for manual tuning across datasets.

A.19 Broader impact

Positive impacts include more expressive graph learning for scientific and molecular applications; negative impacts are indirect and depend on downstream use, including possible misuse in sensitive decision-making if graph classifiers are deployed without validation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions: walk-induced lifted states, structure-aware encoding, sparse/local FWL-style aggregation, theory, and empirical evaluation. These are matched by the method, theory, BREC, OGB, and TU results. The claims are scoped to the default sparse/local model and to the compared baselines.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations of sparse/local higher-order designs, especially poor CFI performance without global structural features, the effect of candidate-set scope, closure depth, and the tradeoff introduced by SPD/PE extensions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The theoretical statements explicitly list assumptions such as injective initial triplet encoding, injective multiset aggregation, injective updates, simple undirected graphs, local closure, complete-bundle filtering, and injective readout. Proofs are provided in Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper gives BREC, OGB, and TU protocols, preprocessing knobs, hyperparameter grids, selected configurations, and reports that source code plus README are provided in supplementary material.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide anonymized source code and instructions in the supplementary material. All benchmark datasets used are public, and the supplementary README describes how to obtain/preprocess them and reproduce the reported experiments

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies BREC relabelings, sample number, training epochs, batch size, optimizer, learning rate, weight decay, threshold, split definitions, OGB scaffold split/evaluator/10 seeds, OGB grid and selected configs, and TU 10-fold CV/hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: OGB results are mean±std over 10 random seeds; TU results are mean±std over 10 folds; BREC uses the Reliable Paired Comparison protocol.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Text in Appendix gives hardware and related runtime/cache information.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public graph benchmarks and does not involve human subjects, private data, scraped personal data, or high-risk model release. The authors have reviewed and followed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: broader impact: positive impacts include more expressive graph learning for scientific and molecular applications; negative impacts are indirect and depend on downstream use, including possible misuse in sensitive decision-making if graph classifiers are deployed without validation.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release pretrained generative models, scraped datasets, human-subject data, or other assets with high misuse risk requiring special safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All benchmark datasets used are public and cited.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We release anonymized code and documentation in the supplementary material, including preprocessing, training, and evaluation instructions.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The work does not involve crowdsourcing or human-subject experiments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work does not involve human subjects, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as an important, original, or non-standard component of the method, experiments, or scientific claims.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.