FROM STARS TO SUBGRAPHS: UPLIFTING ANY GNN WITH LOCAL STRUCTURE AWARENESS

Lingxiao Zhao
Carnegie Mellon University
lingxiao@cmu.edu

Wei Jin
Michigan State University
jinwei2@msu.edu

Leman Akoglu
Carnegie Mellon University
lakoglu@andrew.cmu.edu

Neil Shah
Snap Inc.
nshah@snap.com

ABSTRACT

Message Passing Neural Networks (MPNNs) are a common type of Graph Neural Network (GNN), in which each node's representation is computed recursively by aggregating representations ("messages") from its immediate neighbors akin to a star-shaped pattern. MPNNs are appealing for being efficient and scalable, however their expressiveness is upper-bounded by the 1st-order Weisfeiler-Lehman isomorphism test (1-WL). In response, prior works propose highly expressive models at the cost of scalability and sometimes generalization performance. Our work stands between these two regimes: we introduce a general framework to uplift *any* MPNN to be more expressive, with limited scalability overhead and greatly improved practical performance. We achieve this by extending local aggregation in MPNNs from star patterns to general subgraph patterns (e.g., k-egonets): in our framework, each node representation is computed as the encoding of a surrounding induced subgraph rather than encoding of immediate neighbors only (i.e. a star). We choose the subgraph encoder to be a GNN (mainly MPNNs, considering scalability) to design a general framework that serves as a wrapper to uplift any GNN. We call our proposed method GNN-AK (GNN As Kernel), as the framework resembles a convolutional neural network by replacing the kernel with GNNs. Theoretically, we show that our framework is strictly more powerful than 1&2-WL, and is not less powerful than 3-WL. We also design subgraph sampling strategies which greatly reduce memory footprint and improve speed while maintaining performance. Our method sets new state-of-the-art performance by large margins for several well-known graph ML tasks; specifically, 0.08 MAE on ZINC, 74.79% and 86.887% accuracy on CIFAR10 and PATTERN respectively.

1 INTRODUCTION

Graphs are permutation invariant, combinatorial structures used to represent relational data, with wide applications ranging from drug discovery, social network analysis, image analysis to bioinformatics (Duvenaud et al., 2015; Fan et al., 2019; Shi et al., 2019; Wu et al., 2020). In recent years, Graph Neural Networks (GNNs) have rapidly surpassed traditional methods like heuristically defined features and graph kernels to become the dominant approach for graph ML tasks.

Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) are the most common type of GNNs owing to their intuitiveness, effectiveness and efficiency. They follow a recursive aggregation mechanism where each node aggregates information from its immediate neighbors repeatedly. However, unlike simple multi-layer feedforward networks (MLPs) which are universal approximators of continuous functions (Hornik et al., 1989), MPNNs cannot approximate all permutation-invariant graph functions (Maron et al., 2019b). In fact, their expressiveness is upper bounded by the first order Weisfeiler-Lehman (1-WL) isomorphism test (Xu et al., 2018). Importantly, researchers have shown that such 1-WL equivalent GNNs are not expressive, or powerful, enough to capture basic

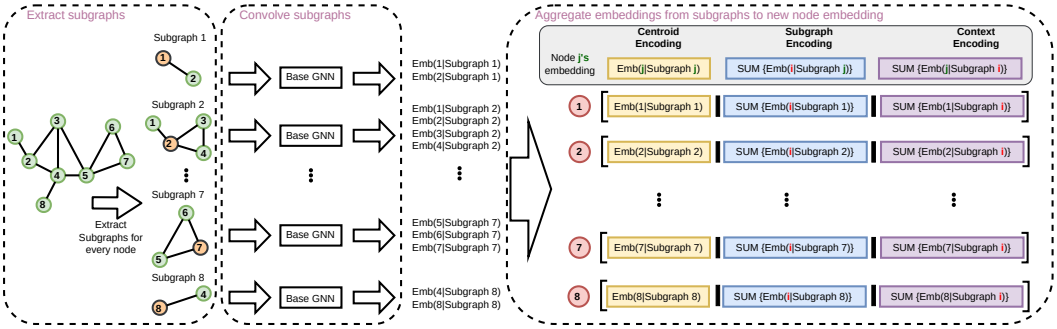

Figure 1: **Shown**: one GNN-AK layer. For each layer, GNN-AK first extracts n (# nodes) subgraphs (e.g 1-hop egonet) rooted at each node, and convolves all subgraphs with a base GNN as kernel, producing multiple rich subgraph-node embeddings of the form $\mathsf{Emb}(i \mid \mathrm{Sub}[j])$ (node i's embedding when applying a GNN kernel on subgraph j). From these, we extract and concatenate three encodings for a given node j: (i) centroid $\mathsf{Emb}(j \mid \mathrm{Sub}[j])$, (ii) subgraph $\sum_i \mathsf{Emb}(i \mid \mathrm{Sub}[j])$ (the output of GNN$(\mathrm{Sub}[j])$, and (iii) context $\sum_i \mathrm{Emb}(j \mid \mathrm{Sub}[i])$. GNN-AK repeats the process for L layers, and sums over all resulting node embeddings to compute the final graph embedding.

structural concepts, i.e., counting motifs such as cycles or triangles (Zhengdao et al., 2020; Arvind et al., 2020) that are shown to be informative for bio- and chemo-informatics (Elton et al., 2019).

The weakness of MPNNs urges researchers to design more expressive GNNs, which are able to discriminate graphs from an isomorphism test perspective; Chen et al. (2019) prove the equivalence between such tests and universal permutation invariant function approximation, which theoretically justifies this line of work. As k-WL is strictly more expressive than 1-WL, many works (Morris et al., 2019; 2020) try to incorporate k-WL in the design of more powerful GNNs, while others approach k-WL expressiveness indirectly from matrix invariant operations (Maron et al., 2019a;b; Keriven & Peyré, 2019) and matrix language perspectives (Balcilar et al., 2021). However, they require O(k)-order tensors to achieve k-WL expressiveness, and thus are not scalable or feasible for application on large, practical graphs. Besides, the bias-variance tradeoff between complexity and generalization (Neal et al., 2018) and the fact that almost all real-world graphs can be distinguished by 1-WL (Cai et al., 1992; Dwivedi et al., 2020) challenge the necessity of developing such extremely expressive models. In a complementary line of work, Loukas (2020a) sheds light on developing more powerful GNNs while maintaining linear scalability, finding that MPNNs can be universal approximators provided that nodes are sufficiently distinguishable. Relatedly, several works propose to add features to make nodes more distinguishable, such as identifiers (Loukas, 2020a), subgraph counts (Bouritsas et al., 2020), distance encoding (Li et al., 2020), and random features (Sato et al., 2021). However, these methods either focus on handcrafted features which lose the premise of automatic learning, or create permutation sensitive features that hurt generalization.

Present Work. Our work stands between the two regimes of extreme expressiveness at the cost of slow and unscalable k-order GNNs, and the limited expressiveness yet high scalability of MPNNs. Specifically, we propose a general framework that serves as a "wrapper" to uplift **any** GNN (mainly MPNNs in practice). We observe that MPNNs' local neighbor aggregation follows a star pattern, where the representation of a node is characterized by applying an injective aggregator function as an encoder to the star subgraph (comprised of the central node and edges to neighbors). We propose a design which naturally generalizes from encoding the star to encoding a more flexibly defined subgraph, and we replace the standard injective aggregator with a GNN: in short, we characterize the new representation of a node by using a GNN to encode a locally induced encompassing subgraph, as shown in Fig.1. This uplifts GNN as a base model in effect by applying it on each *subgraph* instead of the whole input graph. Interestingly, this generalization has a close connection to Convolutional Neural Networks (CNN) in computer vision: like the CNN that convolves image patches with a kernel to compute new pixel embeddings, our designed wrapper convolves subgraphs with a GNN to generate new node embeddings. Hence, we name our approach GNN-AK (GNN As Kernel).

We show theoretically that GNN-AK is strictly more powerful than 1&2-WL with any MPNN as base model, and is not less powerful than 3-WL with PPGN (Maron et al., 2019a) used. We also give sufficient conditions under which GNN-AK can successfully distinguish two non-isomorphic graphs.

Given this increase in expressive power, we discuss careful implementation strategies for GNN-AK, which allow us to carefully leverage multiple modalities of information from subgraph encoding. As a result, GNN-AK induces a constant factor overhead in memory. To amplify our method's practicality, we further develop a subgraph sampling strategy inspired by Dropout (Srivastava et al., 2014) to drastically reduce this overhead (1-3$\times$ in practice) without hurting performance. We conduct extensive experiments on 4 simulation datasets and 5 well-known real-world graph classification & regression benchmarks (Dwivedi et al., 2020; Hu et al., 2020), to show significant and consistent practical benefits of our approach across different MPNNs and datasets. Specifically, GNN-AK sets new state-of-the-art performance on ZINC, CIFAR10, and PATTERN – for example, on ZINC we see a relative error reduction of 60.3%, 50.5%, and 39.4% for base model being GCN (Kipf & Welling, 2017), GIN (Xu et al., 2018), and (a variant of) PNA (Corso et al., 2020) respectively. To summarize, our contributions are listed as follows:

- **A General GNN-AK Framework.** We propose GNN-AK (GNN As Kernel), a general framework which uplifts any GNN by encoding local subgraph structure with a GNN.
- **Theoretical Findings.** We show that GNN-AK's expressiveness is strictly better than 1&2-WL, and is not less powerful than 3-WL. We analyze sufficient conditions for successful discrimination.
- **Effective and Efficient Realization.** We present an effective GNN-AK implementation to fully exploit all node embeddings within a subgraph. We design efficient online subgraph sampling and extraction methods to mitigate memory and runtime overhead while maintaining performance.
- **Experimental Results.** We show strong empirical results, demonstrating both expressivity improvements as well as practical performance gains where we achieve new state-of-the-art performance on several graph-level benchmarks.

Our implementation is easy-to-use, and directly accepts any GNN from PyG (Fey & Lenssen, 2019) for plug-and-play use. See code at `https://github.com/GNNAsKernel/GNNAsKernel`.

2 Related work

Improving Expressiveness of GNNs: Several works other than those mentioned in Sec.1 tackle expressive GNNs. Murphy et al. (2019); Zhengdao et al. (2020) achieve universality by summing permutation-sensitive functions across a combinatorial number of permutations, limiting feasibility. Dasoulas et al. (2020) adds node indicators to make them distinguishable, but at the cost of an invariant model, while Vignac et al. (2020) further addresses the invariance problem, but at the cost of quadratic time complexity. Corso et al. (2020) generalizes MPNN's default sum aggregator, but is still limited by 1-WL. Beani et al. (2021) generalizes spatial and spectral aggregation with $>$1-WL expressiveness, but using expensive eigendecomposition. Recently, Bodnar et al. (2021) introduce MPNNs over simplicial complexes, which shares similar expressiveness as GNN-AK. Azizian & Lelarge (2021) surveys GNN expressiveness work.

Leveraging Substructures in Learning: Exploiting subgraph information in GNNs is not new; in fact, k-WL achieves expressiveness by considering all k node subgraphs. Several works (Monti et al., 2018; Lee et al., 2019) exploit motif information within aggregation, and others (Bouritsas et al., 2020) augment MPNN features with subgraph/motif counts. MixHop (Abu-El-Haija et al., 2019) directly aggregates k-hop information by using adjacency matrix powers, ignoring neighbor connections. Towards a meta-learning goal, G-meta (Huang & Zitnik, 2020) applies GNNs on rooted subgraphs around each node to compute node representations. Nikolentzos et al. (2020) also proposes k-hop GNN which derives node representations by encoding the surrounding k-egonet, but unlike GNN-AK, they only propose to use a final root node embedding. Tahmasebi & Jegelka (2020) theoretically justifies subgraph convolution with GNN by showing its ability in counting substructures, but does not propose a concrete realization. Zhengdao et al. (2020) also computes a node representation by encoding the local subgraph, however using non-scalable relational pooling. Sandfelder et al. (2021) can be viewed as a special case of GNN-AK, which only computes a context embedding (one of three types of embeddings GNN-AK uses) and SGC (Wu et al., 2019) as the subgraph encoder, and only be studied on node-level tasks.

Other Connections: GNN-AK has a similar convolutional structure as CNN, and in fact historically many spatial GNNs are inspired by CNN; see Wu et al. (2020) for a detailed survey. The non-Euclidean nature of graphs makes such generalizations non-trivial. MoNet (Monti et al., 2017) introduces pseudo-coordinates for nodes, while PatchySAN (Niepert et al., 2016) learns to order

and truncate neighboring nodes for convolution purposes. However, both methods aim to mimic the formulation of the CNN without admitting the inherent difference between graphs and images. In contrast, GNN-AK, generalizes CNN to graphs with a base GNN kernel, similar to how a CNN kernel encodes image patches. GNN-AK also shares connections with two variants of k-WL test algorithms: depth-k 1-dim WL (Cai et al., 1992; Weisfeiler, 1976) and deep WL (Arvind et al., 2020). The former recursively applies 1-WL to all size-k subgraphs, with slightly weaker expressiveness than k-WL, and the latter reduces the number of such subgraphs for the k-WL test. Instead of working on all O(n^k) size-k subgraphs, we keep linear scalability by only applying 1-WL-equivalent MPNNs to O(n) rooted subgraphs.

3 The General Framework and Theory

We first introduce our setting and formalisms. Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with node features $\mathbf{x}_i \in \mathbb{R}^d, \forall i \in \mathcal{V}$. We consider graph-level problems where the goal is to classify/regress a target y_G by learning a graph-level representation $\mathbf{h}_G$. Let $\mathcal{N}_k(v)$ be the set of nodes in the k-hop egonet rooted at node v. $\mathcal{N}(v) = \mathcal{N}_1(v) \backslash v$ denotes the immediate neighbors of node v. For $\mathcal{S} \subseteq \mathcal{V}$, let $G[\mathcal{S}]$ be the induced subgraph: $G[\mathcal{S}] = (\mathcal{S}, \{(i,j) \in \mathcal{E} | i \in \mathcal{S}, j \in \mathcal{S}\})$. Then $G[\mathcal{N}_k(v)]$ denotes the k-hop egonet rooted at node v. We also define $Star(v) = (\mathcal{N}_1(v), \{(v,j) \in \mathcal{E} | j \in \mathcal{N}(v)\})$ be the induced star-like subgraph around v.

Before presenting GNN-AK, we highlight the insights in designing GNN-AK and driving the expressiveness boost. **Insight 1: Generalizing star to subgraph.** In MPNNs, every node aggregates information from its immediate neighbors following a star pattern. Consequently, MPNNs fail to distinguish any non-isomorphic regular graphs where all stars are the same, since all nodes have the same degree. Even simply generalizing star to the induced, 1-hop egonet considers connections among neighbors, enabling distinguishing regular graphs. **Insight 2: Divide and conquer.** When two graphs are non-isomorphic, there exists a subgraph where this difference is captured (see Figure 3). Although a fixed-expressiveness GNN may not distinguish the two original graphs, it may distinguish the two *smaller* subgraphs, given that the required expressiveness for successful discrimination is proportional to graph size (Loukas, 2020b). As such, GNN-AK divides the harder problem of encoding the whole graph to smaller and easier problems of encoding its subgraphs, and "conquers" the encoding with the base GNN.

3.1 From Stars to Subgraphs

We first take a close look at MPNNs, identifying their limitations and expressiveness bottleneck. MPNNs repeatedly update each node's embedding by aggregating embeddings from their neighbors a fixed number of times (layers) and computing a graph-level embedding $\mathbf{h}_G$ by global pooling. Let $\mathbf{h}_v^{(l)}$ denote the l-th layer embedding of node v. Then, MPNNs compute $\mathbf{h}_G$ by

$$\mathbf{h}_v^{(l)} = \phi^{(l)}\left(\mathbf{h}_v^{(l)}, f^{(l)}\left(\{\mathbf{h}_u^{(l-1)} | u \in \mathcal{N}(v)\}\right)\right) \quad l = 1, ..., L \quad ; \quad \mathbf{h}_G = \text{POOL}(\{\mathbf{h}_v^{(L)} | v \in \mathcal{V}\}) \quad (1)$$

where $\mathbf{h}_i^{(0)} = \mathbf{x}_i$ is the original features, L is the number of layers, and $\phi^{(l)}$ and $f^{(l)}$ are the l-th layer update and aggregation functions. $\phi^{(l)}$, $f^{(l)}$ and POOL vary among different MPNNs and influence their expressiveness and performance. MPNNs achieve maximum expressiveness (1-WL) when all three functions are injective (Xu et al., 2018).

MPNNs' expressiveness upper bound follows from its close relation to the 1-WL isomorphism test. Similar to MPNNs which repeatedly aggregate self and neighbor representations, at t-th iteration, for each node v, 1-WL test aggregates the node's own label (or color) $c_v^{(t)}$ and its neighbors' labels $\{c_u^{(t)} | u \in \mathcal{N}(v)\}$, and hashes this multi-set of labels $\left\{c_v^{(t)}, \{c_u^{(t)} | u \in \mathcal{N}(v)\}\right\}$ into a new, compressed label $c_v^{(t+1)}$. 1-WL outputs the set of all node labels $\left\{c_v^{(T)} | v \in \mathcal{V}\right\}$ as G's fingerprint, and decides two graphs to be non-isomorphic as soon as their fingerprints differ.

The hash process in 1-WL outputs a new label $c_v^{(t+1)}$ that uniquely characterizes the star graph $Star(v)$ around v, i.e. two nodes u, v are assigned different compressed labels only if $Star(u)$ and $Star(v)$ differ. Hence, it is easy to see that when two non-isomorphic unlabeled (i.e., all nodes have the same label) d-regular graphs have the same number of nodes, 1-WL cannot distinguish them. This failure limits the expressiveness of 1-WL, but also identifies its bottleneck: the star is not

distinguishing enough. Instead, we propose to generalize the star $Star(v)$ to subgraphs, such as the egonet $G[\mathcal{N}_1(v)]$ and more generally k-hop egonet $G[\mathcal{N}_k(v)]$. This results in an improved version of 1-WL which we call *Subgraph-1-WL*. Formally,

Definition 3.1 (**Subgraph-1-WL**). Subgraph-1-WL generalizes the 1-WL graph isomorphism test algorithm by replacing color refinement (at iteration t) by $c_v^{(t+1)} = \text{HASH}(Star^{(t)}(v))$ with $c_v^{(t+1)} = \text{HASH}(G^{(t)}[\mathcal{N}_k(v)]), \forall v \in \mathcal{V}$ where $\text{HASH}(\cdot)$ is an injective function on graphs.

Note that an injective hash function for star graphs is equivalent to that for multi-sets, which is easy to derive (Zaheer et al., 2017). In contrast, Subgraph-1-WL must hash a general subgraph , where an injective hash function for graphs is non-trivial (as hard as graph isomorphism). Thus, we derive a variant called Subgraph-1-WL* by using a weaker choice for $\text{HASH}(\cdot)$ – specifically, 1-WL. Effectively, we *nest* 1-WL inside Subgraph-1-WL. Formally,

Definition 3.2 (**Subgraph-1-WL***). Subgraph-1-WL* is a less expressive variant of Subgraph-1-WL where $c_v^{(t+1)} = \text{1-WL}(G^{(t)}[\mathcal{N}_k(v)])$.

We further transfer Subgraph-1-WL to neural networks, resulting in GNN-AK whose expressiveness is upper bounded by Subgraph-1-WL. The natural transformation with maximum expressiveness is to replace the hash function with a universal subgraph encoder of $G[\mathcal{N}_k(v)]$, which is non-trivial as it implies solving the challenging graph isomorphism problem in the worst case. Analogous to using 1-WL as a weaker choice for $\text{HASH}(\cdot)$ inside Subgraph-1-WL*, we can use use any GNN (most practically, MPNN) as an encoder for subgraph $G[\mathcal{N}_k(v)]$. Let $G^{(l)}[\mathcal{N}_k(v)] = G[\mathcal{N}_k(v)|\mathbf{H}^{(l)}]$ be the attributed subgraph with hidden features $\mathbf{H}^{(l)}$ at the l-th layer. Then, GNN-AK computes $\mathbf{h}_G$ by

$$\mathbf{h}_v^{(l)} = \text{GNN}^{(l)}\Big(G^{(l)}[\mathcal{N}_k(v)]\Big) \quad l = 1, ..., L \quad ; \quad \mathbf{h}_G = \text{POOL}(\{\mathbf{h}_v^{(L)} | v \in \mathcal{V}\}) \tag{2}$$

Notice that GNN-AK acts as a "wrapper" for any base GNN (mainly MPNN). This uplifts its expressiveness as well as practical performance as we demonstrate in the following sections.

3.2 Theory: Expressiveness Analysis

We next theoretically study the expressiveness of GNN-AK, by investigating the expressiveness of Subgraph-1-WL. We first establish that GNN-AK and Subgraph-1-WL have the same expressiveness under certain conditions. A GNN is able to distinguish two graphs if its embeddings for two graphs are not identical. A GNN is said to have the same expressiveness as a graph isomorphism test when for any two graphs the GNN outputs different embeddings if and only if (iff) the isomorphism test deems them non-isomorphic. We give following theorems with proof in Appendix.

Theorem 1. *When the base model is an MPNN with sufficient number of layers and injective ϕ, f and POOL functions shown in Eq. (1), and MPNN-AK has an injective POOL function shown in Eq. (2), then MPNN-AK is as powerful as Subgraph-1-WL*.*

See Appendix.A.1 for proof. A more general version of Theorem 1 is that GNN-AK is as powerful as Subgraph-1-WL iff base GNN of GNN-AK is as powerful as the HASH function of Subgraph-1-WL in distinguishing subgraphs, following the same proof logic. The Theorem implies that we can characterize expressiveness of GNN-AK through studying Subgraph-1-WL and Subgraph-1-WL*.

Theorem 2. *Subgraph-1-WL* is strictly more powerful than 1&2-WL.*

A direct corollary from Theorem 1&2 is as follows, which is empirically verified in Table 1.

Corollary 2.1. *When MPNN is 1-WL expressive, MPNN-AK is strictly more powerful than 1&2-WL.*

Theorem 3. *When HASH$(\cdot)$ is 3-WL expressive, Subgraph-1-WL is no less powerful than 3-WL, and can discriminate some graphs for which 3-WL fails.*

A direct corollary from Theorem 1&3 is as follows, which is empirically verified in Table 1.

Corollary 3.1. *PPGN-AK can distinguish some 3-WL-failed non-isomorphic graphs.*

Theorem 4. *For any $k \geq 3$, there exists a pair of k-WL-failed graphs that cannot be distinguished by Subgraph-1-WL even with injective HASH$(\cdot)$ when d-hop egonets are used with $d \leq 4$.*

Theorem 4 is proven (Appendix.A.4) by observing that with limited d all rooted subgraphs of two non-isomorphic graphs from $CFI(k)$ family (Cai et al., 1992) are isomorphic, i.e. local rooted subgraph is not enough to capture the "global" difference. This opens a future direction of generalizing rooted subgraph to general subgraph (as in k-WL) while keeping number of subgraphs in $\text{O}(|\mathcal{V}|)$.

Conjecture 1 (sufficient conditions)**.** *For two non-isomorphic graphs G, H, Subgraph-1-WL can successfully distinguish them if: 1) for any node reordering there exists some k-egonets $G[\mathcal{N}_k(u)]$ $H[\mathcal{N}_k(v)]$ ($u \in \mathcal{V}_G, v \in \mathcal{V}_H$) that are non-isomorphic; 2) HASH$(\cdot)$ is discriminative enough that HASH$(G[\mathcal{N}_k(u)]) \neq$ HASH$(H[\mathcal{N}_k(v)])$.*

This implies subgraph size should be large enough to capture difference, but not larger which requires more expressive base model (Loukas, 2020a). We empirically verify Conj. 1 in Table 2.

4 Concrete Realization

Next, we present concrete designs to effectively and efficiently realize GNN-AK in practice. Specifically, we present (1) three types of encodings that fully exploit all intermediate node embeddings across rooted subgraphs; (2) a subgraph pooling design to incorporate distance-to-centroid, readily computed during subgraph extraction; and (3) a random-walk based rooted subgraph extraction for graphs with small diameter to reduce memory footprint of k-hop egonets that can reach almost entire graph even with a small k. We conclude this section with time and space complexity analysis.

4.1 Design and Realization

Three Types of Encodings. Let $G = (\mathcal{V}, \mathcal{E})$ be the input graph with $N = |\mathcal{V}|$, $G^{(l)}[\mathcal{N}_k(v)]$ be the k-hop egonet rooted at node $v \in \mathcal{V}$ in which $\mathbf{h}_u^{(l)}$ denotes node u's hidden representation for $u \in \mathcal{N}_k(v)$ at the l-th layer of GNN-AK. To simplify notation, we use $\mathrm{Sub}^{(l)}[v]$ instead of $G^{(l)}[\mathcal{N}_k(v)]$ to generally indicate the the attribute-enriched induced subgraph for v. We consider all intermediate node embeddings across rooted subgraphs, specifically, $\mathsf{Emb}(i \mid \mathrm{Sub}^{(l)}[j])$ denotes node i's embedding when base GNN$^{(l)}$ is applied on $\mathrm{Sub}^{(l)}[j]$, for every $j \in \mathcal{V}$ and every $i \in \mathrm{Sub}^{(l)}[j]$. Note that the base GNN can have multiple graph convolutional layers, and Emb refers to the node embeddings at the last layer before global pooling POOL$_{\text{GNN}}$ that generates subgraph-level encoding. Accordingly we can formally rewrite Eq. (2) as

$$\mathbf{h}_v^{(l)|\text{Subgraph}} = \text{GNN}^{(l)}\Big(\mathrm{Sub}^{(l)}[v]\Big) := \text{POOL}_{\text{GNN}^{(l)}}\Big(\big\{\mathsf{Emb}(i \mid \mathrm{Sub}^{(l)}[v]) \mid i \in \mathcal{N}_k(v)\big\}\Big) \tag{3}$$

The encoding in Eq. (3) is the default and referred to as subgraph encoding, as it encodes the rooted subgraph $\mathrm{Sub}^{(l)}[v]$. Typical choices of POOL$_{\text{GNN}^{(l)}}$ are SUM and MEAN.

We observe that subgraph encoding alone does not fully exploit all information side $\sum_{v \in \mathcal{V}} |\mathcal{N}_k(v)|$ intermediate embeddings, and propose two additional encodings, namely centroid and context.

$$\mathbf{h}_v^{(l)|\text{Centroid}} := \mathsf{Emb}(v \mid \mathrm{Sub}^{(l)}[v]) \tag{4}$$

$$\mathbf{h}_v^{(l)|\text{Context}} := \text{POOL}_{\text{Context}}\Big(\big\{\mathsf{Emb}(v \mid \mathrm{Sub}^{(l)}[j]) \mid \forall j \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{5}$$

We remark that these three different encodings contain non-redundant information, which is empirically verified in Table 6 : for node v, subgraph encoding captures the default encoding of rooted subgraph around v; centroid encoding only takes the root of the subgraph, which is exactly the representation of applying base GNN on original graph when base GNN only has 1 layer; and context encoding captures views of node v from different subgraph points-of-view, i.e. representation in "context" of others. Overall, we realize Eq. (2) in practice by fusing all three encodings as

$$\mathbf{h}_v^{(l)} = \text{FUSE}\Big(\mathbf{h}_v^{(l)|\text{Centroid}},\ \mathbf{h}_v^{(l)|\text{Subgraph}},\ \mathbf{h}_v^{(l)|\text{Context}}\Big) \tag{6}$$

where FUSE is concatenation or sum. We illustrate in Figure 1 the l-th layer of GNN-AK.

Subgraph Extraction and Distance-to-centroid Feature. GNN-AK extracts the rooted subgraph for every node and records the node and edge mapping between original graph and $|\mathcal{V}|$ rooted subgraphs. The subgraph extraction can be preprocessed offline to maximize speed, while in practice we implement it online without much overhead, thanks to fast k-hop propagation with complexity $O(k|\mathcal{E}|)$. Notably, along with k-hop propagation for extracting k-hop egonet, the distance to centroid (or root node) within each subgraph is readily recorded at no additional cost and can be used to augment node features, which has been shown to help expressiveness (Li et al., 2020) theoretically. Therefore, GNN-AK uses the distance-to-centroid by default in two ways: (1) augmenting hidden representation $\mathbf{h}_v^{(l)}$ in Eq. (6) by concatenating it with the encoding of distance-to-centroid; (2) using it to gate the subgraph and context encodings before POOL$_{\text{Subgraph}}$ and POOL$_{\text{Context}}$, the latter inspired by the hypothesis that embeddings far from v contribute differently than those close to v.

To formalize the gate mechanism guided by distance-to-centroid (D2C), let $\mathbf{d}_{i|j}^{(l)}$ be the encoding of distance from node i to j at l-th layer[1]. Applying gate mechanism changes Eq. (5) to

$$\mathbf{h}_v^{(l)|\text{Context}} := \text{POOL}_{\text{Context}}\Big(\big\{\text{Sigmoid}(\mathbf{d}_{v|j}^{(l)}) \odot \textsf{Emb}(v \mid \text{Sub}^{(l)}[j]) \mid \forall j \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{7}$$

where $\odot$ denotes element-wise multiplication. Similar changes apply to Eq. (3).

We provide ablation study for the proposed node encodings and D2C feature in Table 6.

Beyond k-egonet Subgraphs. The k-hop egonet (or k-egonet) is a natural choice in our framework, but can be too large when the input graph's diameter is small, as in social networks (Kleinberg, 2000), or when the graph is dense. To limit subgraph size, we also design a random-walk based subgraph extractor. Specifically, to extract a subgraph rooted at node v, we perform a fixed-length random walk starting at v, resulting in visited nodes $\mathcal{N}_{rw}(v)$ and their induced subgraph $G[\mathcal{N}_{rw}(v)]$. In practice, we use adaptive random walks as in Grover & Leskovec (2016). To reduce randomness, we use multiple truncated random walks and union the visited nodes as $\mathcal{N}_{rw}(v)$. Moreover, we employ online subgraph extraction during training that re-extracts subgraphs at every epoch, which further alleviates the effect of randomness via regularization.

4.2 Complexity Analysis

Let us assume k-egonets as rooted subgraphs, and an MPNN as base model. For each graph $G = (\mathcal{V}, \mathcal{E})$, the subgraph extraction takes $O(k|\mathcal{E}|)$ runtime complexity, and outputs $|\mathcal{V}|$ subgraphs, which collectively can be represented as a union graph $\mathcal{G}_\cup = (\mathcal{V}_\cup, \mathcal{E}_\cup)$ with $|\mathcal{V}|$ disconnected components, where $|\mathcal{V}_\cup| = \sum_{v\in\mathcal{V}} |\mathcal{N}_k(v)|$ and $|\mathcal{E}_\cup| = \sum_{v\in\mathcal{V}} |\mathcal{E}_{G[\mathcal{N}_k(v)]}|$. GNN-AK can be viewed as applying base GNN on the union graph. Assuming the base GNN has $O(|\mathcal{V}| + |\mathcal{E}|)$ runtime and memory complexity at forward pass, GNN-AK has $O(|\mathcal{V}_\cup| + |\mathcal{E}_\cup|)$ runtime and memory cost. For rooted subgraphs of size s, GNN-AK induces an $O(s)$ factor overhead over the base model.

5 Improving Scalability: SubgraphDrop

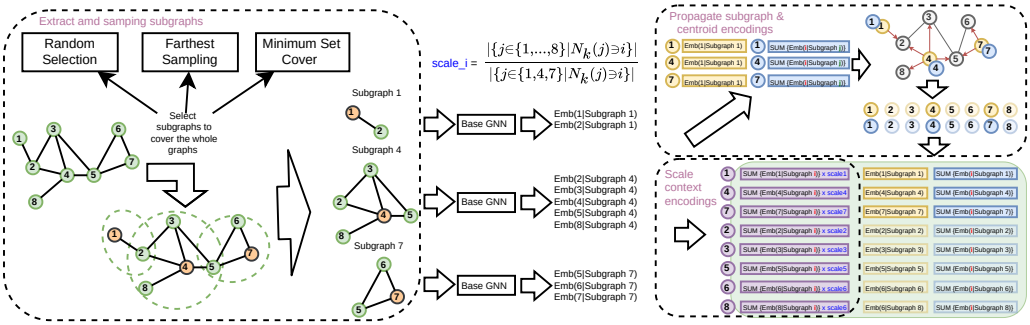

Figure 2: GNN-AK-S with SubgraphDrop used in training. GNN-AK-S first extracts subgraphs and subsamples $m \ll |\mathcal{V}|$ subgraphs to cover each node at least R times with multiple strategies. The base GNN is applied to compute all intermediate node embeddings in selected subgraphs. Context encodings are scaled to match evaluation. Subgraph and centroid encodings initially only exist for root nodes of selected subgraphs, and are propagated to estimate those of other nodes.

The complexity analysis reveals that GNN-AK introduces a constant factor overhead (subgraph size) in runtime and memory over the base GNN. Subgraph size can be naturally reduced by choosing smaller k for k-egonet, or by ranking and truncating visited nodes in a random walk setting. However limiting to very small subgraphs tends to hurt performance as we empirically show in Table 5. Here, we present a different subsampling-based approach that carefully selects only a subset of the $|\mathcal{V}|$ rooted subgraphs. Further more, inspired from Dropout (Srivastava et al., 2014), we only drop subgraphs during training while still use all subgraphs when evaluation. Novel strategies are designed specifically for three type of encodings to eliminate the estimation bias between training and evaluation. We name it SubgraphDrop for dropping subgraphs during training. SubgraphDrop significantly reduces memory overhead while keeping performance nearly the same as training with

[1] D2C does not change across layers but is encoded differently from layer to layer.

all subgraphs. We first present subgraph sampling strategies, then introduce the designs of aligning training and evaluation. Fig. 2 provides a pictorial illustration.

5.1 SUBGRAPH SAMPLING STRATEGIES

Intuitively, if u, v are directly connected in G, the subgraphs $G[\mathcal{N}_k(u)]$ and $G[\mathcal{N}_k(v)]$ share a large overlap, and may contain redundancy. With this intuition, we aim to sample only $m \ll |\mathcal{V}|$ minimally redundant subgraphs to reduce memory overhead. We propose three fast sampling strategies that select subgraphs to evenly cover the whole graph, where each node is covered by $\approx R$ (redundancy factor) selected subgraphs; R is a hyperparameter used as a sampling stopping condition. Then, GNN-AK-S (with sampling) has roughly R times the overhead of base model, ($R \leq 3$ in practice).

- **Random sampling** selects subgraphs randomly until every node is covered $\geq R$ times.
- **Farthest sampling** selects subgraphs iteratively, starting at a random one and greedily selecting each subsequent one whose root node is farthest w.r.t. shortest path distance from those of already selected subgraphs, until every node is covered $\geq R$ times.
- **Min-set-cover sampling** initially selects a subgraph randomly, and follows the greedy minimum set cover algorithm to iteratively select the subgraph containing the maximum number of uncovered nodes, until every node is covered $\geq R$ times.

We remark that all subsampling strategies are randomized and fast, which are both desired characteristics for an online Dropout-like sampling during training.

5.2 TRAINING WITH SUBGRAPHDROP

Although dropping redundant subgraphs greatly reduces overhead, it still loses information. Thus, as in Dropout (Srivastava et al., 2014), we only "drop" subgraphs during training while still using all of them during evaluation. Randomness in sampling strategies enforces that selected subgraphs differ across training epochs, preserving most information due to amortization. On the other hand, it makes it difficult for the three encodings during training to align with full-mode evaluation. Next, we propose an alignment procedure for each type of encoding.

Subgraph and Centroid Encoding in Training. Let $\mathcal{S} \subseteq \mathcal{V}$ be the set of root nodes of the selected subgraphs. When sampling during training, subgraph and centroid encoding can only be computed for nodes $v \in \mathcal{S}$, following Eq. (3) and Eq. (4), resulting incomplete subgraph encodings $\{\mathbf{h}_v^{(l)|\text{Subgraph}} | v \in \mathcal{S}\}$ and centroid encodings $\{\mathbf{h}_v^{(l)|\text{Centroid}} | v \in \mathcal{S}\}$. To estimate uncomputed encodings of $u \in \mathcal{V} \setminus \mathcal{S}$, we propose to *propagate* encodings from $\mathcal{S}$ to $\mathcal{V} \setminus \mathcal{S}$. Formally, let $k_{max} = \max_{u \in \mathcal{V} \setminus \mathcal{S}} \text{Dist}(u, \mathcal{S})$ where $\text{Dist}(u, \mathcal{S}) = \min_{v \in \mathcal{S}} \text{ShortestPathDistance}(u, v)$. Then we partition $\mathcal{U} = \mathcal{V} \setminus \mathcal{S}$ into $\{\mathcal{U}_1, ..., \mathcal{U}_{k_{max}}\}$ with $\mathcal{U}_d = \{u \in \mathcal{U} | \text{Dist}(u, \mathcal{S}) = d\}$. We propose to spread vector encodings of $\mathcal{S}$ out iteratively, i.e. compute vectors of $\mathcal{U}_d$ from $\mathcal{U}_{d-1}$. Formally, we have

$$\mathbf{h}_u^{(l)|\text{Subgraph}} = \text{Mean}\big(\{\mathbf{h}_v^{(l)|\text{Subgraph}} | v \in \mathcal{U}_{d-1}, (u, v) \in \mathcal{E}\}\big) \quad \text{for} \quad d = 1, 2 \dots k_{max}, \forall u \in \mathcal{U}_d, \tag{8}$$

$$\mathbf{h}_u^{(l)|\text{Centroid}} = \text{Mean}\big(\{\mathbf{h}_v^{(l)|\text{Centroid}} | v \in \mathcal{U}_{d-1}, (u, v) \in \mathcal{E}\}\big) \quad \text{for} \quad d = 1, 2 \dots k_{max}, \forall u \in \mathcal{U}_d, \tag{9}$$

Context Encoding in Training. Following Eq. (5), context encodings can be computed for every node $v \in \mathcal{V}$ as each node is covered at least R times during training with SubgraphDrop. However when $\text{POOL}_{\text{Context}}$ is $\text{SUM}(\cdot)$, the scale of $\mathbf{h}_v^{(l)|\text{Context}}$ is smaller than the one in full-mode evaluation. Thus, we scale the context encodings up to align with full-mode evaluation. Formally,

$$\mathbf{h}_v^{(l)|\text{Context}} = \frac{|\{j \in \mathcal{V} | \mathcal{N}_k(j) \ni v\}|}{|\{j \in \mathcal{S} | \mathcal{N}_k(j) \ni v\}|} \times \text{SUM}\Big(\big\{\mathsf{Emb}(v \mid \text{Sub}^{(l)}[j]) \mid \forall j \in \mathcal{S} \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{10}$$

When $\text{POOL}_{\text{Context}}$ is $\text{MEAN}(\cdot)$, the context encoding is computed without any modification.

6 EXPERIMENTS

In this section we 1) empirically verify the expressiveness benefit of GNN-AK on 4 simulation datasets; 2) show GNN-AK boosts practical performance significantly on 5 real-world datasets; 3) demonstrate the effectiveness of SubgraphDrop; 4) conduct ablation studies of concrete designs.

6.1 SETUP

Simulation Datasets. 1) EXP (Abboud et al., 2020) contains 600 pairs of 1&2-WL failed graphs that are splited into two classes where each graph of a pair is assigned to two different classes. 2) SR25 (Balcilar et al., 2021) has 15 strongly regular graphs (3-WL failed) with 25 nodes each.

SR25 is translated to 15 way classification problem with the goal of mapping each graph into a different class. 3)Substructure counting (i.e. triangle, tailed triangle, star and 4-cycle) problems on random graph dataset (Zhengdao et al., 2020). 4) Graph property regression (i.e. connectedness, diameter, radius) tasks on random graph dataset (Corso et al., 2020). All simulation datasets used to empirically verify the expressiveness of GNN-AK.

Real-world Datasets. ZINC-12K, CIFAR10, PATTER from Benchmarking GNNs (Dwivedi et al., 2020) and MolHIV, and MolPCBA from Open Graph Benchmark (Hu et al., 2020).

Baselines. We use GCN (Kipf & Welling, 2017), GIN (Xu et al., 2018), PNA* [2] (Corso et al., 2020), and 3-WL powerful PPGN (Maron et al., 2019a) directly, which also server as base model of GNN-AK to see its general uplift effect. GatedGCN (Dwivedi et al., 2020), DGN (Beani et al., 2021), PNA (Corso et al., 2020), GSN(Bouritsas et al., 2020), HIMP (Fey et al., 2020) are referenced directly from literature for real-world datasets comparison.

Hyperparameter and model configuration. To reduce the search space, we search hyperparameters in a two-phase approach: First, we search common ones (hidden size from [64, 128], number of layers L from [2,4,5,6], (sub)graph pooling from [SUM, MEAN] for each dataset using GIN based on validation performance, and fix it for any other GNN and GNN-AK. While hyperparameters may not be optimal for other GNN models, the evaluation is fair as there is no benefit for GNN-AK. Next, we search GNN-AK exclusive ones (encoding types) over validation set using GIN-AK and keep them fixed for other GNN-AK. We use a 1-layer base model, 3-hop egonets for GNN-AK, with excpetions that CIFAR10 uses 2-hop egonet due to memory constraint; PATTERN uses random walk based subgraph with walk length=10 and repeated 5 times due to high density stochastic block graph. For GNN-AK-S, $R = 3$ is set as default. We use farthest sampling for molecular datasets ZINC, MolHIV, and MolPCBA; to speed up further random sampling is used for CIFAR10 whose graphs are k-NN graphs; min-set-cover sampling is used for PATTERN to adapt random walk based subgraph. We use Batch Normalization and ReLU activation in all models. For optimization we always use Adam with learning rate 0.001.

6.2 Empirical Verification of Expressiveness

Table 1: Simulation dataset performance: *GNN-AK boosts any base GNN across tasks, empirically verifying expressiveness lift*. (ACC: accuracy, MAE: mean absolute error, OOM: out of memory)

Method	EXP (ACC)	SR25 (ACC)	Counting Substructures (MAE)				Graph Properties ($\log_{10}$(MAE))		
			Triangle	Tailed Tri.	Star	4-Cycle	IsConnected	Diameter	Radius
GCN	50%	6.67%	0.4186	0.3248	0.1798	0.2822	-1.7057	-2.4705	-3.9316
GCN-AK	**100%**	6.67%	0.0137	0.0134	0.0174	0.0183	-2.6705	-3.9102	-5.1136
GIN	50%	6.67%	0.3569	0.2373	0.0224	0.2185	-1.9239	-3.3079	-4.7584
GIN-AK	**100%**	6.67%	0.0123	0.0112	0.0150	0.0126	-2.7513	-3.9687	-5.1846
PNA*	50%	6.67%	0.3532	0.2648	0.1278	0.2430	-1.9395	-3.4382	-4.9470
PNA*-AK	**100%**	6.67%	0.0118	0.0138	0.0166	0.0132	-2.6189	-3.9011	-5.2026
PPGN	100%	6.67%	0.0089	0.0096	0.0148	0.0090	-1.9804	-3.6147	-5.0878
PPGN-AK	100%	**100%**	OOM	OOM	OOM	OOM	OOM	OOM	OOM

Table 1 presents the results on simulation datasets. All GNN-AK variants perform perfectly on $1\&2$-WL failed EXP where all MPNNs also fail, which verified that GNN-AK is strictly more powerful than 1-WL. Moreover, PPGN-AK reaches perfect accuracy on 3-WL failed SR25, where PPGN fails as it is 3-WL powerful. Similarly, GNN-AK consistently boosts all MPNNs for substructure and graph property prediction tasks (PPGN-AK is OOM as it is quadratic in input size).

Table 2: PPGN-AK expressiveness on SR25.

PPGN's #L	1-hop egonet			2-hop egonet		
Iterations	1	2	3	1	2	3
1	26.67%	100%	100%	26.67%	26.67%	46.67%
2	33.33%	100%	100%	26.67%	26.67%	53.33%
3	26.67%	100%	100%	33.33%	26.67%	53.33%

In Table 2 we look into PPGN-AK's performance on SR25 as a function of k-egonets ($k\in[1, 2]$), as well as the number of PPGN layers and iterations for PPGN-AK. We find that at least 2 inner layers is needed with 1-egonet subgraphs to achieve top performance. With 2-egonets more inner lay-

[2] PNA* is a variant of PNA that changes from using degree to scale embeddings to encoding degree and concatenate to node embeddings. This eliminates the need of computing average degree of datasets in PNA.

ers helps, although performance is sub-par, attributed to PPGN's disability to distinguish larger (sub)graphs, aligning with Conjecture 1 (Sec. 3.2) that GNN-AK needs large enough subgraphs to capture difference, but not larger which requires more expressive base model.

6.3 Comparing with SOTA and Generality

Having studied expressiveness tasks, we turn to performance on real-world datasets, as shown in Table 3. We observe similar performance lifts across all datasets and base GNNs (we omit PPGN due to scalability), demonstrating our framework's generality. Remarkably, GNN-AK sets new SOTA performance for several benchmarks – ZINC, CIFAR10, and PATTERN – with a relative error reduction of 60.3%, 50.5%, and 39.4% for base model being GCN, GIN, and PNA* respectively.

Table 3: Real-world dataset performance: *GNN-AK achieves SOTA performance for ZINC-12K, CIFAR10, and PATTERN.* (OOM: out of memory, −: missing values from literature)

Method	ZINC-12K (MAE)	CIFAR10 (ACC)	PATTERN (ACC)	MolHIV (ROC)	MolPCBA (AP)
GatedGCN	0.363 ± 0.009	69.37 ± 0.48	84.480 ± 0.122	–	–
HIMP	0.151 ± 0.006	–	–	0.7880 ± 0.0082	–
PNA	0.188 ± 0.004	70.47 ± 0.72	86.567 ± 0.075	0.7905 ± 0.0132	0.2838 ± 0.0035
DGN	0.168 ± 0.003	72.84 ± 0.42	86.680 ± 0.034	$\mathbf{0.7970 \pm 0.0097}$	0.2885 ± 0.0030
GSN	0.115 ± 0.012	–	–	0.7799 ± 0.0100	–
GCN	0.321 ± 0.009	58.39 ± 0.73	85.602 ± 0.046	0.7422 ± 0.0175	0.2385 ± 0.0019
GCN-AK	0.127 ± 0.004	72.70 ± 0.29	$\mathbf{86.887 \pm 0.009}$	0.7928 ± 0.0101	0.2846 ± 0.0002
GIN	0.163 ± 0.004	59.82 ± 0.33	85.732 ± 0.023	0.7881 ± 0.0119	0.2682 ± 0.0006
GIN-AK	$\mathbf{0.080 \pm 0.001}$	72.19 ± 0.13	86.850 ± 0.057	0.7961 ± 0.0119	$\mathbf{0.2930 \pm 0.0044}$
PNA*	0.140 ± 0.006	73.11 ± 0.11	85.441 ± 0.009	0.7905 ± 0.0102	0.2737 ± 0.0009
PNA*-AK	0.085 ± 0.003	OOM	OOM	0.7880 ± 0.0153	0.2885 ± 0.0006
GCN-AK-S	0.127 ± 0.001	71.93 ± 0.47	86.805 ± 0.046	0.7825 ± 0.0098	0.2840 ± 0.0036
GIN-AK-S	0.083 ± 0.001	72.39 ± 0.38	86.811 ± 0.013	0.7822 ± 0.0075	0.2916 ± 0.0029
PNA*-AK-S	0.082 ± 0.000	$\mathbf{74.79 \pm 0.18}$	86.676 ± 0.022	0.7821 ± 0.0143	0.2880 ± 0.0012

6.4 Scaling up by Subsampling

Table 4: Resource analysis of SubgraphDrop.

Dataset		GIN-AK-S R=1	R=2	R=3	R=4	R=5	GIN-AK	GIN
ZINC-12K	MAE	0.1216	0.0929	0.0846	0.0852	0.0854	0.0806	0.1630
	Runtime (S)	10.8	11.2	12.0	12.4	12.5	9.4	6.0
	Memory (MB)	392	811	1392	1722	1861	1911	124
CIFAR10	ACC	71.68	72.07	72.39	72.20	72.32	72.19	59.82
	Runtime (S)	80.7	89.1	100.5	110.9	119.7	241.1	55.0
	Memory (MB)	2576	4578	6359	8716	10805	30296	801

In some cases, GNN-AK's overhead leads to OOM, especially for complex models like PNA* that are resource-demanding. Sampling with SubgraphDrop enables training using practical resources. Notably, GNN-AK-S models, shown at the end of Table 3, do not compromise and can *even improve* performance as compared to their non-sampled counterpart, in alignment with Dropout's benefits Srivastava et al. (2014). Thus, SubgraphDrop has two-fold advantages: reduced time & memory footprint, and improved generalization. Next we evaluate resource-savings, specifically on ZINC-12K and CIFAR10. Table 4 shows that GIN-AK-S with varying R provides an effective handle to trade off resources with performance. Importantly, the rate in which performance decays with smaller R is much lower than the rates at which runtime and memory decrease.

6.5 Ablation Study

We present ablation results on various structural components of GNN-AK. Table 5 shows the performance of GIN-AK for varying size egonets with k. We find that larger subgraphs tend to yield improvement, although runtime-performance trade-off may vary by dataset. Notably, simply 1-egonets are enough for CIFAR10 to uplift performance of the base GIN considerably.

Table 5: Effect of various k-egonet size.

Ablation of GIN-AK	ZINC-12K	CIFAR10
GIN	0.163 ± 0.004	59.82 ± 0.33
$k = 1$	0.147 ± 0.006	71.37 ± 0.28
$k = 2$	0.120 ± 0.005	72.19 ± 0.13
$k = 3$	0.080 ± 0.001	OOM

Table 6: Effect of various encodings

Ablation of GIN-AK	ZINC-12K	CIFAR10
Full	0.080 ± 0.001	72.19 ± 0.13
w/o Subgraph encoding	0.086 ± 0.001	67.76 ± 0.29
w/o Centroid encoding	0.084 ± 0.003	72.20 ± 0.96
w/o Context encoding	0.088 ± 0.003	69.25 ± 0.30
w/o Distance-to-Centroid	0.085 ± 0.001	71.91 ± 0.22

Next Table 6 illustrates the added benefit of various node encodings. Compared to the full design, eliminating any of the subgraph, centroid, or context encodings (Eq.s (3)–(5)) yields notably inferior results. Encoding without distance awareness is also subpar. These justify the design choices in our framework and verify the practical benefits of our design.

7 Conclusion

Our work introduces a new, general-purpose framework called GNN-As-Kernel (GNN-AK) to uplift the expressiveness of any GNN, with the key idea of employing a base GNN as a kernel on induced subgraphs of the input graph, generalizing from the star-pattern aggregation of classical MPNNs. Our approach provides an expressiveness and performance boost, while retaining practical scalability of MPNNs—a highly sought-after middle ground between the two regimes of scalable yet less-expressive MPNNs and high-expressive yet practically infeasible and poorly-generalizing k-order designs. We theoretically studied the expressiveness of GNN-AK, as well as provided a concrete design, introducing SubgraphDrop during training for shrinking runtime and memory footprint. Extensive experiments on both simulated and real-world benchmark datasets empirically justified that GNN-AK (i) uplifts base GNN expressiveness for multiple base GNN choices (e.g. over $1/2$-WL for MPNNs, and over 3-WL for PPGN), (ii) which translates to performance gains with SOTA results on several graph-level benchmarks, (iii) while retaining scalability to practical graphs.

REFERENCES

Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On weisfeiler-leman invariance: subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113:42–59, 2020.

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=lxHgXYN4bwl`.

Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Dominique Beani, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Lió. Directional graph networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 748–758, 2021.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lió, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 1026–1037, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

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.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13260–13271, 2020.

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 2126–2132. International Joint Conferences on Artificial Intelligence Organization, 2020.

Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

David K Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, 2015.

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

Daniel C Elton, Zois Boukouvalas, Mark D Fuge, and Peter W Chung. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design & Engineering*, 4(4):828–849, 2019.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

M. Fey, J. G. Yuen, and F. Weichert. Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workhop*, 2020.

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

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. *Advances in Neural Information Processing Systems*, 33, 2020.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32:7092–7101, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pp. 163–170, 2000.

John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. Graph convolutional networks with motif-based attention. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 499–508, 2019.

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

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=B1l2bp4YwS`.

Andreas Loukas. How hard is to distinguish graphs with graph neural networks? In *Advances in Neural Information Processing Systems*, 2020b.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pp. 2156–2167, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5115–5124, 2017.

Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pp. 225–228. IEEE, 2018.

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*, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33, 2020.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, 2018.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023. PMLR, 2016.

Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

Dylan Sandfelder, Priyesh Vijayan, and William L Hamilton. Ego-gnns: Exploiting ego structures in graph neural networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8523–8527. IEEE, 2021.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 333–341. SIAM, 2021.

Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7912–7921, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Behrooz Tahmasebi and Stefanie Jegelka. Counting substructures with higher-order graph neural networks: Possibility and impossibility results. *arXiv preprint arXiv:2012.03174*, 2020.

Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems*, 2020.

Boris Weisfeiler. On construction and identification of graphs. In *LECTURE NOTES IN MATHEMATICS*. Citeseer, 1976.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.

Chen Zhengdao, Chen Lei, Villar Soledad, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 2020.

A APPENDIX

A.1 PROOF OF THEOREM 1

Proof. (Xu et al., 2018) proved that with sufficient number of layers and all injective functions, MPNN is as powerful as 1-WL. Then Eq.2 outputs different vectors for two graphs iff Subgraph-1-WL* encodes different labels with $c_v^{(t+1)} = \text{1-WL}(G^{(t)}[\mathcal{N}_k(v)])$. With POOL in Eq.2 also to be injective, MPNN-AK outputs different vectors iff Subgraph-1-WL* outputs different fingerprints for two graphs. Then MPNN-AK is as powerful as Subgraph-1-WL*. □

A.2 PROOF OF THEOREM 2

Proof. We first prove that if two graphs are identified as isomorphic by Subgraph-1-WL*, they are also determined as isomorphic by 1-WL. Then we present a pair of non-isomorphic graphs that can be distinguished by Subgraph-1-WL* but not by 1-WL. Together these two imply that Subgraph-1-WL* is strictly more powerful than 1-WL. Comparing with 2-WL can be concluded from the fact that 1-WL and 2-WL are equivalent in expressiveness (Maron et al., 2019a). In the proof we use 1-hop egonet subgraphs for Subgraph-1-WL*.

Assume graphs G and H have the same number of nodes (otherwise easily determined as non-isomorphic) and are two non-isomorphic graphs but Subgraph-1-WL* determines them as isomorphic. Then for any iteration t, set $\left\{\text{1-WL}(G^{(t)}[\mathcal{N}_1(v)])|v \in \mathcal{V}_G\right\}$ is the same as set $\left\{\text{1-WL}(H^{(t)}[\mathcal{N}_1(v)])|v \in \mathcal{V}_H\right\}$. Then there existing an ordering of nodes $v_1^G, ..., v_n^G$ and $v_1^H, ..., v_N^H$ with $N = |\mathcal{V}_G| = |\mathcal{V}_H|$, such that for any node order $i = 1, ..., N$, $\text{1-WL}(G^{(t)}[\mathcal{N}_1(v_i^G)]) = \text{1-WL}(H^{(t)}[\mathcal{N}_1(v_i^H)])$. This implies that structure $G[\mathcal{N}_1(v_i^G)]$ and $H[\mathcal{N}_1(v_i^H)]$ are not distinguishable by 1-WL. Hence $Star(v_i^G)$ and $Star(v_i^H)$ are hashed to the same label otherwise the 1-WL that includes the hashed result of $Star(v_i^G)$ and $Star(v_i^H)$ can also distinguish $G[\mathcal{N}_1(v_i^G)]$ and $H[\mathcal{N}_1(v_i^H)]$. Then for any iteration t and any node with order i, $\text{HASH}\left(Star(v_i^{G^{(t)}})\right) = \text{HASH}\left(Star(v_i^{H^{(t)}})\right)$ implies that 1-WL fails in distinguishing G and H. In fact if we replace the 1-WL hashing function in Subgraph-1-WL* to a stronger version $\text{HASH}\left(\left\{\text{HASH}(Star(v^{G^{(t)}})), \text{1-WL}(G^{(t)}[\mathcal{N}_1(v)])\right\}\right)$, this directly implies the above statement.

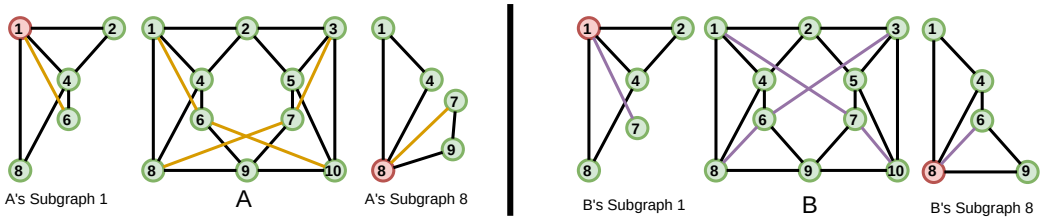

Figure 3: Two 4-regular graphs that cannot be distinguished by 1-WL. Colored edges are the difference between two graphs. Two 1-hop egonets are visualized while all other rooted egonets are ignored as they are same across graph A and graph B.

In Figure 3, two 4-regular graphs are presented that cannot be distinguished by 1-WL but can be distinguished by Subgraph-1-WL*. We visualize the 1-hop egonets that are structurally different among graphs A and B. It's easy to see that A's egonet $A[\mathcal{N}_1(1)]$ and B's egonet $B[\mathcal{N}_1(1)]$ can be distinguished by 1-WL, as degree distribution is not the same. Hence, A and B can be distinguished by Subgraph-1-WL*. □

A.3 PROOF OF THEOREM 3

Proof. We prove by showing a pair of 3-WL failed can be distinguished by Subgraph-1-WL, assuming HASH$(\cdot)$ is 3-WL discriminative. Figure 4 shows two strongly regular graphs that can not be distinguished by 3-WL (any strongly regular graphs are not distinguishable by 3-WL (Arvind et al., 2020)), along with their 1-hop egonet rooted subgraphs. Notice that all 1-hop egonet rooted

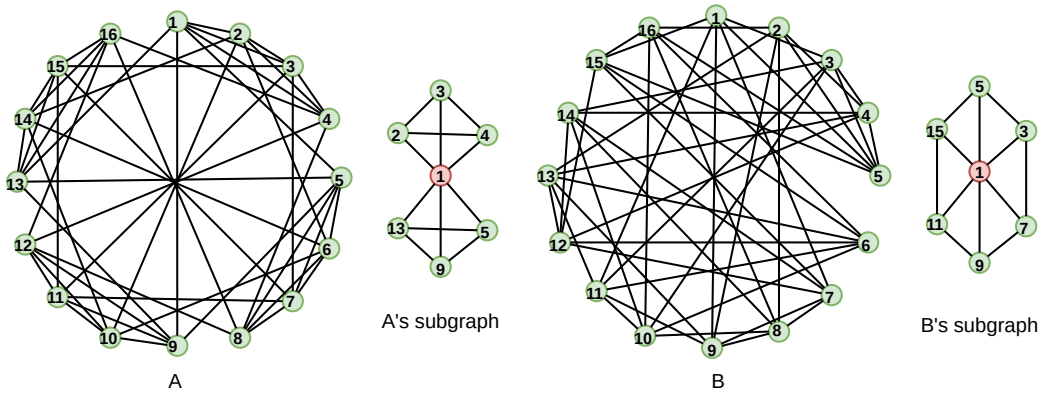

Figure 4: Two non-isomorphic strongly regular graphs that cannot be distinguished by 3-WL.

subgraphs in A (also in B) are the same, resulting that Subgraph-1-WL can successfully distinguish A and B if HASH can distinguish the showed two 1-hop egonets. Notice that these two egonets are regular graphs that can be distinguished by 3-WL. Hence, when HASH$(\cdot)$ is 3-WL expressive, Subgraph-1-WL can distinguish A and B. □

A.4 Proof of Theorem 4

Proof. For any $k \geq 3$, Cai et al. (1992) provided a scheme to construct two k-WL failed non-isomorphic graphs. Specifically, it first constructs one graph A from a base graph G replacing every degree-k node in graph G to a carefully designed separator graph with size $2^{k-1} + k$, and then rewire two edges in A to its non-isomorphic counter graph B. We refer to Cai et al. (1992) and Douglas (2011) for clear description of the construction process. It's known that the separator graph has diameter $= 4$, and the rewiring of constructing its non-isomorphic counter graph has picked two edges that cannot be included by any k-hop egonets with $k \leq 4$. This leads to the fact that all rooted k-hop egonets are isomorphic to each other. Hence, no matter what HASH$(\cdot)$ function is used, the function always output same value for any rooted subgraphs in A and B. Therefore, for any k, Subgraph-1-WL cannot distinguish the two graphs A and B that are k-WL-failed. □