A. Weisfeiler-Lehman Test

The universality of a GNN is based on its ability to embed two non-isomorphic graphs to distinct points in the target feature space. A model which can distinguish all pairs of non-isomorphic graphs is a universal approximator. Since it is not known if the graph isomorphism problem can be solved in polynomial time or not, this problem is neither NP-complete nor P, but NP-intermediate (Takapoui & Boyd, 2016). One of the oldest but prominent polynomial approach is the Weisfeiler-Lehman Test (abbreviated WL-test) which gives sufficient but not enough evidence. WL test can be extended by taking into account higher order of node tuple within the iterative process. These extensions are denoted as k-WL test, where k is equal to the order of the tuple. It is important to mention that an higher order of tuple leads to a better ability to distinguish two non-isomorphic graphs (with the exception for $k = 2$) (Arvind et al., 2020).

The 1-WL test, known as vertex coloring, starts with the given initial color of nodes if available. Otherwise all nodes are colored with the same color ($H_v^{(0)} = 1$). Then, colors are updated by the following iteration:

$$H_v^{(t+1)} = \sigma\left(H_v^{(t)} \mid \left\{H_u^{(t)} : u \in \mathcal{N}(v)\right\}\right), \quad (7)$$

where $H_v^{(t)}$ is the color of vertex v at iteration t, $\mathcal{N}(v)$ is the set of neighbours of vertex v, $\mid$ represents the concatenation operator and $\{.\}$ is the order invariant multiset[3]. In order to avoid the new color of vertex become bigger after each iteration due to the concatenation operation and to keep the color description simple, the recoloring $\sigma(\cdot)$ function is applied after each iteration. It assigns a new simple color identifier to the any newly created color. The test is performed in parallel for two graphs. The iterative process is stopped when the color histograms are kept unchanged between two consecutive iterations. The color histograms associated to the compared graphs are examined. If in any iteration the histograms are different, we can conclude that the graphs are not isomorphic. However, the opposite conclusion can not be drawn if color histograms are equal as two same histograms may be computed even for non-isomorphic graphs.

Higher order WL tests use the same algorithm while their color update schema is slightly different. The 2-WL test uses second order tuple of nodes (all ordered pairs of nodes), thus it needs $\mathbf{H} \in \mathbb{R}^{n\times n}$ matrix, where the initial color set has two more colors than initial vertex colors as defined by:

$$\mathbf{H}_{v,u}^{(0)} = \begin{cases} H_v^{(0)} & \text{if } v = u \\ edge & \text{if } u \in \mathcal{N}(v) \\ nonedge & \text{if } u \notin \mathcal{N}(v) \end{cases} \quad (8)$$

Then, the iteration process is applied through the following schema where $[n]$ is the set of node identifiers.

$$\mathbf{H}_{v,u}^{(t+1)} = \sigma\left(\mathbf{H}_{v,u}^{(t)} \mid \left\{\mathbf{H}_{v,k}^{(t)} : k \in [n]\right\} \mid \left\{\mathbf{H}_{k,u}^{(t)} : k \in [n]\right\}\right), \quad (9)$$

Although for $k \geq 2$, $(k+1)$-WL is more powerful than (k)-WL, it is not true for $k = 1$, thus 2-WL (Eq.(9)) is no more powerful than 1-WL (Eq.(7)) (Maron et al., 2019a). To clarify this point, Folkore WL (FWL) test is defined such that 1-WL=1-FWL, but for $k \geq 2$, we have $(k+1)$-WL $\approx (k)$-FWL (Maron et al., 2019a). The iteration process of 2-FWL is given by the following equation;

$$\mathbf{H}_{v,u}^{(t+1)} = \sigma\left(\mathbf{H}_{v,u}^{(t)} \mid \left\{\left(\mathbf{H}_{v,k}^{(t)} | \mathbf{H}_{k,u}^{(t)}\right) : k \in [n]\right\}\right), \quad (10)$$

In the literature, there are different interpretations of the order of the WL test. Some papers use WL test order to denote the iteration given by Eq.(7) and Eq.(9) (Morris et al., 2019; Maron et al., 2019a) but some others such as (Abboud et al., 2020; Arvind et al., 2020; Takapoui & Boyd, 2016) use FWL order under the name of WL. In this paper, we explicitly mention both WL and FWL equivalent such as 3-WL (or 2-FWL) to alleviate ambiguities.

B. Proofs of Theorems

B.1. Theorem.1

Proof. All these methods can be written in Eq.(1) by different convolution matrices C. The main idea of the proof is that as long as convolution matrices C can be explained by operations from the enriched set $\mathcal{L}_1^+$ (Remark 4), Eq.(1) also can be explained by operations from $\mathcal{L}_1^+$ as well. Thus these methods cannot produce any sentence out of $\mathcal{L}_1^+$. As a consequence, their expressive power is not more than 1-WL test. To provide a proof, the mentioned methods' convolution matrices have to be expressed using operations from $\mathcal{L}_1^+$.

GCN uses $C = (D+I)^{-0.5}(A+I)(D+I)^{-0.5}$ where D is the diagonal degree matrix (Kipf & Welling, 2017) in Eq.(1). $(D+I)^{-0.5}$ can be expressed as $(D+I)^{-0.5} = diag(f(A\mathbf{1}+\mathbf{1}))$, where $f(x) = x^{-0.5}$ is element-wise operation on vector x. $A+I$ can also be written $A + diag(\mathbf{1})$. When we merge these equations, we get $C = diag(f(A\mathbf{1}+\mathbf{1}))(A + diag(\mathbf{1}))diag(f(A\mathbf{1}+\mathbf{1}))$. The convolution support C is then written using operations from $\mathcal{L}_1^+$.

In the literature, GraphSage method was proposed to sample neighborhood and aggregate the neighborhood contribution by the mean operator or LSTM in (Hamilton et al., 2017). Since we restrict the method using full sampling and mean aggregator, we can define GraphSage by the general framework given by Eq.(1) with two convolution supports which are the identity matrix $C^{(1)} = I$ and the row normalized adjacency matrix $C^{(2)} = D^{-1}A$. These convolution supports

[3] It is generally implemented by stacking all colors in the set and sorting them alphabetically

can also be expressed by operations from $\mathcal{L}_1^+$, by observing that $C^{(1)} = diag(\mathbf{1})$ and $C^{(2)} = diag(f(A\mathbf{1}))A$, where $f(x) = x^{-1}$ elementwise operation on vector x.

GIN (Xu et al., 2019) uses a convolution support $C = A + I\epsilon$ in Eq.(1) which is followed by a custom number of MLP layers. Each of these layers correspond to a convolution support that can by expressed as $C_{mlp} = I$ in Eq.(1). Finally, these convolution supports can be written thanks to operations from $\mathcal{L}_1^+$. $C = A + \epsilon \times diag(\mathbf{1})$ and $C_{mlp} = diag(\mathbf{1})$.

GAT (Veličković et al., 2018) can be expressed in Eq.(1) by the convolution support designed by $C_{v,u} = m(H_v, H_u)/\sum_{k\in\tilde{\mathcal{N}}(v)} m(H_v, H_k)$, where $\tilde{\mathcal{N}}(v)$ is the self-connection added neighborhood of v and $m(.)$ is any trainable model. If we write the trainable model $m(.)$ as a sum of each node such as $m(H_v, H_u) = f_1(H_v) + f_2(H_u)$, we can define an intermediate matrix $B = diag(f_1(H))(A + I) + (A + I)diag(f_2(H))$. Finally the GAT convolution support can be written by $C = diag((B\mathbf{1})^{-1})B$ using all operations included within the operation set $\mathcal{L}_1^+$. □

B.2. Theorem.2

Proof. Chebnet (Defferrard et al., 2016) uses desired number k of convolution supports in Eq.(1). As long as these convolutions can be written by operations in $\mathcal{L}_1^+$, we can conclude that Chebnet is no more powerful than 1-WL test. But if at least one convolution cannot be explained in $\mathcal{L}_1^+$, we can say it is more powerful than 1-WL test.

Chebnet's convolution supports are $C^{(1)} = I$, $C^{(2)} = 2L/\lambda_{\max} - I$, $C^{(k)} = 2C^{(2)}C^{(k-1)} - C^{(k-2)}$. The first support can always be written thanks to an operation from $\mathcal{L}_1$ since $C^{(1)} = diag(\mathbf{1})$. Both normalized and combinatorial graph Laplacian can also be written as $L = diag(A\mathbf{1}) - A$ or $L = diag(\mathbf{1}) - diag(f(A\mathbf{1}))Adiag(f(A\mathbf{1}))$ where $f(x) = x^{-1/2}$ elementwise operation on vector x. If λ_{max} for both graphs are the same, we can use a constant $\alpha = 2/\lambda_{max}$. The second convolution support can then be written as $C^{(2)} = \alpha \times L - diag(\mathbf{1})$. It is then expressed by means of operations from $\mathcal{L}_1^+$. Other convolution supports $C^{(k)} = 2C^{(2)}C^{(k-1)} - C^{(k-2)}$ are created by matrix multiplication and subtraction of previous supports which can all be expressed by mean of operations from $\mathcal{L}_1^+$. Thus, if the maximum eigenvalues of tested graphs Laplacians are the same, Chebnet is not more powerful than 1-WL.

However, if the maximum eigenvalues are not the same, $C^{(2)}$ cannot be expressed with the help of the constant value α. It means that different coefficients should be used for each graph. For two tested graphs G and H, we can write second kernel of Chebnet as $C_G^{(2)} = \alpha_G \times L_G - diag(\mathbf{1})$ and $C_H^{(2)} = \alpha_H \times L_H - diag(\mathbf{1})$. If these two graphs are 1-WL equivalent, any sentence build on $\mathcal{L}_1^+$ applied on these graph is equivalent as well. For instance, we can use the sentences of $e(X) = \mathbf{1}^\top X\mathbf{1}$ with operation in $\mathcal{L}_1^+$. The output of the sentence should be same such $e(L_G) = e(L_H)$ yields $\mathbf{1}^\top L_G\mathbf{1} = \mathbf{1}^\top L_H\mathbf{1}$. If we assume that Chebnet cannot separate these two graphs, we can calculate one layer ChebNet's output by second support with the same sentence and they should be the same such $e(C_G^{(2)}) = e(C_H^{(2)})$ yields $\alpha_G\mathbf{1}^\top L_G\mathbf{1} = \alpha_H\mathbf{1}^\top L_H\mathbf{1}$. Last equation has contradiction to the previous one as long as the maximum eigenvalues are not same (i.e $\alpha_G \neq \alpha_H$) and graphs are not regular (i.e $\mathbf{1}^\top L_G\mathbf{1} > 0$ and $\mathbf{1}^\top L_H\mathbf{1} > 0$ for normalized laplacian). This contradiction says that assumption is wrong, so one layer Chebnet's second support can distinguish 1-WL equivalent graphs whose maximum eigenvalues are not same and graphs are not regular with the same degree. □

Since the graph laplacians are positive semi-definite, it always yields $\mathbf{1}^\top L_G\mathbf{1} \geq 0$ and $\mathbf{1}^\top L_H\mathbf{1} \geq 0$ and they are zero as long as the graphs are regular with the same degree. Thus, if we add smallest positive scalar value on the diagonal of the laplacian such $L \leftarrow L + \epsilon I$, we get rid of the necessity that graphs must be non-regular. So Chebnet become more powerful and will be able to distinguish all 1-WL equivalent regular graphs whose maximum eigenvalues are different. Considering the graph8c task, we have seen that classic ChebNet could not distinguish 44 pairs where there are 312 1-WL equivalent pairs. If we use $L \leftarrow L + 0.01I$, the number of undistinguished pairs of graph decreased from 44 to 19, where 19 undistinguished pairs are all 1-WL equivalent and have exact the same maximum eigenvalues. On the other hand, original Chebnet was not able to distinguish 44-19=25 graphs pairs whose maximum eigenvalues are different but all of them are regular thus $\mathbf{1}^\top L\mathbf{1} = 0$.

B.3. Theorem.3

Proof. The number of 3-star patterns can be determined by $\sum_v \binom{d(v)}{3}$ where $d(v)$ is the degree of vertex v for undirected simple graphs (Pinar et al., 2017). Using $f(x) = \frac{x!}{(x-3)!3!}$ as a function that operates on each element of a given vector x, we can calculate the number of 3-star patterns in a given adjacency matrix A by $\mathbf{1}^\top f(A\mathbf{1})$ using operations in $\mathcal{L}_1^+$. According to the universal approximation theory of multi layer perceptron (Hornik et al., 1989), if we have enough layers, we can implement $f(.)$ as an MLP in our model. □

B.4. Theorem.4

Proof. The number of triangles can be determined by using trace operator as $1/6 \times tr(A^3)$ (Harary & Manvel, 1971) which can be written by means of operations from $\mathcal{L}_2^+$.

Number of 4-cycles is determined by $1/8 \times (tr(A^4) + tr(A^2) - 2\mathbf{1}^\top A^2\mathbf{1})$ (Harary & Manvel, 1971) which can be

written by means of operations from $\mathcal{L}_2^+$. □

B.5. Theorem.5

Proof. If $t(v)$ denotes the number triangles including vertex v and $d(v)$ denotes the degree of vertex v, the number of tailed triangles can be found by $\sum_v t(v).(d(v)-2)$ for simple undirected graphs (Pinar et al., 2017). Every node in a triangle has two closed walks of length 3. Thus, $t(v) = \frac{(A^3)_{v,v}}{2}$. It yields the number of tailed triangles can be found by $\frac{1}{2} \times \mathbf{1}^\top (A^3 \odot diag(A\mathbf{1}-2))\mathbf{1}$. The computation of $t(v)$ which involves the element-wise multiplication can be written with operations from $\mathcal{L}_3^+$. □

B.6. Theorem.6

Proof. Since the sentences in $ML(\mathcal{L}_1)$ produce a scalar value which can be reached in the graph readout layer as a sum thanks to $\mathbf{1}^\top H^{(l_{end})}$, we need to show that the MPNN can produce all possible vectors in $\mathcal{L}_1$ on the last node representation layer. Since $H^{(0)} = \mathbf{1}$, the output of the first layer consists of linear combination of $[\mathbf{1}, A\mathbf{1}]$ because, in this case, the third term of the sum is just $\mathbf{1} \circ \mathbf{1} = \mathbf{1}$. On the second layer, the representation consists of a linear transformation of 4 different vectors $[\mathbf{1}, A\mathbf{1}, A^2\mathbf{1}, A\mathbf{1} \circ A\mathbf{1}]$. We can notice that these 4 vectors are the all possible vectors that $\mathcal{L}_1$ can produce up to the second level. The $diag$ operator can produce other outputs if we apply $diag(A\mathbf{1}).diag(A\mathbf{1})\mathbf{1} = A\mathbf{1} \circ A\mathbf{1}$. Because $diag(\mathbf{1}) = I$ cannot change anything if we use it any other expressions. Another selection would be $A.diag(A\mathbf{1})\mathbf{1} = A^2\mathbf{1}$ and last option gives $diag(A\mathbf{1})A\mathbf{1} = A\mathbf{1} \circ A\mathbf{1}$. So up to $l = 2$ the proof is true. Then, we follow an inductive reasoning and assume that in the k-th layer, Eq.(2) produces all possible vectors $(h_1, \dots h_n)$ in $\mathcal{L}_1$ and we show that it is true for $k+1$-th layer as well. In the $k+1$-th layer, the first term of the sum keeps $h_1, \dots h_n$. The second term produces $Ah_1, \dots Ah_n$. Finally, the term of the sum produces all pairs of element-wise multiplication such as $h_1 \circ h_1, h_1 \circ h_2, \dots h_n \circ h_n$. These are the all vectors that the language $\{., \mathbf{1}, diag\}$ can produce using one extra A and/or $diag$ operator. The transpose operator is neglected because the adjacency matrix is symmetric. Furthermore, since at the readout layer these vectors are to be summed up, their order or the fact that they are transposed or not does not matter.

Beside, it was also shown that $diag(.)$ operator can be implemented by element-wise multiplication of vectors in (Geerts, 2020) in Proposition 8.1. □

B.7. Theorem.7

Proof. If the given function is $\Phi(\lambda)$, it can be written by power series using the Maclaurin expansion as follows:

$$\Phi(\lambda) = \frac{\Phi(0)}{0!}\lambda^0 + \frac{\Phi'(0)}{1!}\lambda^1 + \frac{\Phi^{(2)}(0)}{2!}\lambda^2 + \dots. \quad (11)$$

Thus, the frequency response can be written by power series with coefficients $\alpha_i = \frac{\Phi^{(i)}(0)}{i!}$. Using these coefficients, the convolution support can be formulated as

$$C = \alpha_0 UIU^\top + \alpha_1 U diag(\lambda)U^\top + \alpha_2 U diag(\lambda)^2 U^\top + \dots \quad (12)$$

Since $UIU^\top = I = L^0$ and $U diag(\lambda)^n U^\top = L^n$, we can reach the final expression:

$$C = \alpha_0 L^0 + \alpha_1 L^1 + \alpha_2 L^2 + \dots \quad (13)$$

The convolution support C is expressed as power series of graph laplacian L as long as all order derivation of frequency response is not zero ($\Phi^{(n)}(0) \neq 0$). Since the selection of the function is based on $exp(.)$ and its derivation is never null, we can conclude that designed convolution support can be written by power series of graph Laplacian. □

C. $\mathcal{L}_1$ Equivalent Graphs

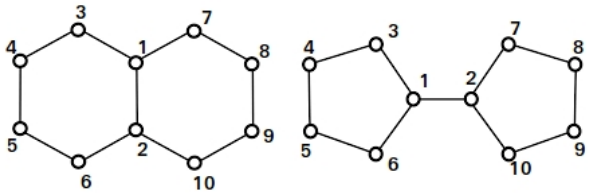

Figure 4. Decalin and Bicyclopentyl graphs are $\mathcal{L}_1$ equivalent and so 1-WL.

Figure 4, shows Decalin and Bicyclopentyl graphs, with a proposed node enumeration. According to these enumerations, their adjacency matrices are A_G and A_H, respectively

$$A_G = \begin{pmatrix} 0&1&1&0&0&0&1&0&0&0\\ 1&0&0&0&0&1&0&0&0&1\\ 1&0&0&1&0&0&0&0&0&0\\ 0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0\\ 0&1&0&0&1&0&0&0&0&0\\ 1&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&1&0&1\\ 0&1&0&0&0&0&0&0&1&0 \end{pmatrix} \text{ and } A_H = \begin{pmatrix} 0&1&1&0&0&1&0&0&0&0\\ 1&0&0&0&0&0&1&0&0&1\\ 1&0&0&1&0&0&0&0&0&0\\ 0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0\\ 1&0&0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&1&0&1\\ 0&1&0&0&0&0&0&0&1&0 \end{pmatrix}$$

Their normalized Laplacian can be calculated by $L = I - D^{-1/2}AD^{-1/2}$ and gives L_G and L_H as follows:

$$L_G = \begin{pmatrix} 1&-0.33&-0.41&0&0&0&-0.41&0&0&0\\ -0.33&1&0&0&0&-0.41&0&0&0&-0.41\\ -0.41&0&1&-0.5&0&0&0&0&0&0\\ 0&0&-0.5&1&-0.5&0&0&0&0&0\\ 0&0&0&-0.5&1&-0.5&0&0&0&0\\ 0&-0.41&0&0&-0.5&1&0&0&0&0\\ -0.41&0&0&0&0&0&1&-0.5&0&0\\ 0&0&0&0&0&0&-0.5&1&-0.5&0\\ 0&0&0&0&0&0&0&-0.5&1&-0.5\\ 0&-0.41&0&0&0&0&0&0&-0.5&1 \end{pmatrix}$$

$$L_H = \begin{pmatrix} 1 & -0.33 & -0.41 & 0 & 0 & -0.41 & 0 & 0 & 0 & 0 \\ -0.33 & 1 & 0 & 0 & 0 & 0 & -0.41 & 0 & 0 & -0.41 \\ -0.41 & 0 & 1 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.5 & 1 & -0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 1 & -0.5 & 0 & 0 & 0 & 0 \\ -0.41 & 0 & 0 & 0 & -0.5 & 1 & 0 & 0 & 0 & 0 \\ 0 & -0.41 & 0 & 0 & 0 & 0 & 1 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.50 & 1 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.50 & 1 & -0.5 \\ 0 & -0.41 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 1 \end{pmatrix}$$

Their second Chebnet convolution supports are $C_G^{(2)} = 2/2L_G - I$ and $C_H^{(2)} = 2/1.8418L_H - I$ because their maximum eigenvalues are 2.0 and 1.8418 respectively. Finally, when computing the output of the first layer by linear activation function without any learning parameters, we obtain $y_G = \mathbf{1}^\top C_G^{(2)}\mathbf{1} = -9.9327$ and $y_H = \mathbf{1}^\top C_H^{(2)}\mathbf{1} = -9.9269$. We observe a slight difference between these two values, which means that Chebnet can project both graphs to the different points, thus it is able to distinguish them.

Since the maximum eigenvalues of graphs Laplacians are different, they are not cospectral as well. It means that they can also be distinguished on the basis of the number closed walks for some lengths which can be determined by trace operator. Indeed, even if up to 4th power of the adjacency matrix, the trace operator gives the same values for both graphs, we can observe that $tr(A_G^5) = 0$ whereas $tr(A_H^5) = 20$. This observation is sufficient to claim that both graphs are not $\mathcal{L}_2$ equivalent.

D. $\mathcal{L}_2$ Equivalent Graphs

Figure 5 shows two non-isomorphic but $\mathcal{L}_2$ equivalent graphs, where vertices are enumerated.

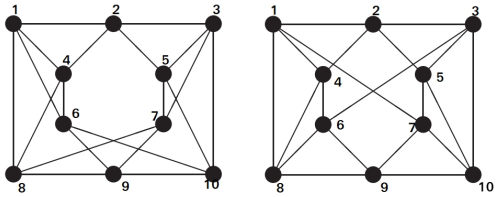

Figure 5. Cospectral and 4-regular graphs from (Van Dam & Haemers, 2003) are $\mathcal{L}_2$ equivalent.

According to these enumerations, their adjacency matrices are the following:

$$A_G = \begin{pmatrix} 0&1&0&1&0&1&0&1&0&0 \\ 1&0&1&1&1&0&0&0&0&0 \\ 0&1&0&0&1&0&1&0&0&1 \\ 1&1&0&0&0&1&0&1&0&0 \\ 0&1&1&0&0&0&1&0&0&1 \\ 1&0&0&1&0&0&0&0&1&1 \\ 0&0&1&0&1&0&0&1&1&0 \\ 1&0&0&1&0&0&1&0&1&0 \\ 0&0&0&0&0&1&1&1&0&1 \\ 0&0&1&0&1&1&0&0&1&0 \end{pmatrix} \text{ and } A_H = \begin{pmatrix} 0&1&0&1&0&0&1&1&0&0 \\ 1&0&1&1&1&0&0&0&0&0 \\ 0&1&0&0&1&1&0&0&0&1 \\ 1&1&0&0&0&1&0&1&0&0 \\ 0&1&1&0&0&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1&1&0 \\ 1&0&0&0&1&0&0&0&1&1 \\ 1&0&0&1&0&1&0&0&1&0 \\ 0&0&0&0&0&1&1&1&0&1 \\ 0&0&1&0&1&0&1&0&1&0 \end{pmatrix}$$

We have seen that their normalized Laplacian eigenvalues are $\lambda_G = \lambda_H = [0, 0.44, 0.61, 0.75, 1.25, 1.25, 1.25, 1.25, 1.56, 1.64]$. Thus, they are cospectral. Considering that for cospectral graphs, the trace of any power of the adjacency matrix which gives the number of closed walks, is the same, we conclude that the trace operator does not help to distinguish these two graphs.

For instance, it can be verified that the trace of the adjacency matrix up to its 5th power is equal: $tr(A_G^2) = tr(A_H^2) = 40$, $tr(A_G^3) = tr(A_H^3) = 48$, $tr(A_G^4) = tr(A_H^4) = 360$, and $tr(A_G^5) = tr(A_H^5) = 920$).

However, the sentence $e(X) = \mathbf{1}^\top((X \odot X^2)^2\mathbf{1})^2$ which implements the element-wise multiplication from $\mathcal{L}_3$ allows to distinguish both graphs. Indeed, the computation of this sentences on A_G and A_H gives $\mathbf{1}^\top((A_G \odot A_G^2)^2\mathbf{1})^2 = 6032$ and $\mathbf{1}^\top((A_H \odot A_H^2)^2\mathbf{1})^2 = 5872$. Thus, these two graphs are not $\mathcal{L}_3$ equivalent (it means not 3-WL or 2-FWL equivalent as well) because the sample sentence can be explained in $\mathcal{L}_3$.

E. $\mathcal{L}_3$ Equivalent Graphs

Strongly regular graphs are known to be 3-WL equivalent and $\mathcal{L}_3$ equivalent as well. Figure 6 shows sample non-isomorphic graphs that are $\mathcal{L}_3$ equivalent.

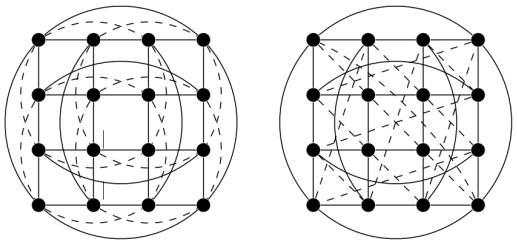

Figure 6. Strongly regular graph pair. 4 × 4-rook's graph and the Shrikhande graph from (Arvind et al., 2020) are $\mathcal{L}_3$ equivalent.

When we enumerate the nodes from the top-left to the bottom-right according to their locations in the Figure 6, their adjacency matrices are the following:

$$A_G = \begin{pmatrix} 0&1&1&1&1&0&0&0&1&0&0&0&1&0&0&0 \\ 1&0&1&1&0&1&0&0&0&1&0&0&0&1&0&0 \\ 1&1&0&1&0&0&1&0&0&0&1&0&0&0&1&0 \\ 1&1&1&0&0&0&0&1&0&0&0&1&0&0&0&1 \\ 1&0&0&0&0&1&1&1&1&0&0&0&1&0&0&0 \\ 0&1&0&0&1&0&1&1&0&1&0&0&0&1&0&0 \\ 0&0&1&0&1&1&0&1&0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&1&1&0&0&0&0&1&0&0&0&1 \\ 1&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0 \\ 0&1&0&0&0&1&0&0&1&0&1&1&0&1&0&0 \\ 0&0&1&0&0&0&1&0&1&1&0&1&0&0&1&0 \\ 0&0&0&1&0&0&0&1&1&1&1&0&0&0&0&1 \\ 1&0&0&0&1&0&0&0&1&0&0&0&0&1&1&1 \\ 0&1&0&0&0&1&0&0&0&1&0&0&1&0&1&1 \\ 0&0&1&0&0&0&1&0&0&0&1&0&1&1&0&1 \\ 0&0&0&1&0&0&0&1&0&0&0&1&1&1&1&0 \end{pmatrix}$$

$$A_H = \begin{pmatrix}
0&1&0&1&1&1&0&0&0&0&0&0&1&0&0&1\\
1&0&1&0&0&1&1&0&0&0&0&0&1&1&0&0\\
0&1&0&1&0&0&1&1&0&0&0&0&0&1&1&0\\
1&0&1&0&1&0&0&1&0&0&0&0&0&0&1&1\\
1&0&0&1&0&1&0&1&1&1&0&0&0&0&0&0\\
1&1&0&0&1&0&1&0&0&1&1&0&0&0&0&0\\
0&1&1&0&0&1&0&1&0&0&1&1&0&0&0&0\\
0&0&1&1&1&0&1&0&1&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&1&0&1&0&1&1&1&0&0\\
0&0&0&0&1&1&0&0&1&0&1&0&0&1&1&0\\
0&0&0&0&0&1&1&0&0&1&0&1&0&0&1&1\\
0&0&0&0&0&0&1&1&1&0&1&0&1&0&0&1\\
1&1&0&0&0&0&0&0&1&0&0&1&0&1&0&1\\
0&1&1&0&0&0&0&0&1&1&0&0&1&0&1&0\\
0&0&1&1&0&0&0&0&0&1&1&0&0&1&0&1\\
1&0&0&1&0&0&0&0&0&0&1&1&1&0&1&0
\end{pmatrix}$$

The eigenvalues of the normalized Laplacian are equal ($\lambda_G = \lambda_H$). Both normalized Laplacians have 3 distinct eigenvalues which are 0, 0.667 and 1.33 with the respective multiplicity of 1, 6 and 9. Thus the graphs are cospectral. Since they are 3-WL equivalent, none of the sentences in $\mathcal{L}_3$ can distinguish these graphs. For instance, we have seen that $\mathbf{1}^\top((A_G \odot A_G^2)^2\mathbf{1})^2 = \mathbf{1}^\top((A_H \odot A_H^2)^2\mathbf{1})^2 = 331776$.

In order to distinguish these two graphs, we need to mimic the 3-FWL (or 4-WL) test which needs a 3-order relationship between nodes. Thus, the adjacencies will be represented by $A_G, A_H \in \mathbb{R}^{16\times16\times16}$. For any 3 nodes there are 3 different pairs and thus $2^3 = 8$ different states representing how these 3 nodes are connected or not. An additional state is used for the tensor diagonal. Thus, there is a total of 9 states. The node tuple is denoted by $A_{i,j,k} \in \{0, \dots, 8\}$. 0 refers to the fact that none of three nodes are connected. 7 refers to the fact that all nodes are mutually connected (triangle). 8 is used for the tensor diagonal elements. We can then define an equivariant 3 dimensional tensor square operator by $(A^2)_{i,j,k} = \sum_s (A_{s,j,k}.A_{i,s,k}.A_{i,j,s})$. By summing all elements of the 3-dimensional squared adjacency where the given adjacency is for instance 0, we can distinguish these two graphs. Indeed, $\sum(A_G^2 \odot (A_G = 0)) = 205632$ whereas $\sum(A_H^2 \odot (A_H = 0)) = 208704$. We can then conclude that these two graphs are not 3-FWL (or 4-WL) equivalent.

F. Result of TU Datasets

Table 5 shows the results of 10-fold cross validation over studied datasets named MUTAG, ENZYMES, PROTEINS and PTC. All these datasets consist of chemical molecules where nodes refer to atoms while edges refer to atomic bonds. For these molecular datasets, node features is a one hot coding of atom types and none of the model use any edge feature even if it exists for MUTAG. In addition to these results, we also provide results on the ENZYMES dataset using extra 18-length continuous features on atoms. Using these continuous features, graph agnostic method MLP performance increases drastically from 30.8% to 70.6%, showing that these continuous features contain at least a part of the structural information. Models were ran for a fixed number of epochs on each fold and we select the epoch where the general accuracy is maximum on the validation set. The test procedure and train/validation split was taken from (Xu et al., 2019).

G. Datasets and Application Details

Table 6 shows the summary of the dataset used in experimental evaluation. The evaluation has been performed on four differents tasks depending on the dataset. These are graph isomorphism (Iso), graph regression (Reg), node regression (NReg) and n-class graph classification task (#-Class). We did not use any edge features even if some were available. All features were defined on nodes. These features were discrete node labels coded by one-hot vectors (#Label) and/or continuous features referred by numbers in Tab. 6. We can notice that some graphs have no feature on nodes.

We get the Graph8c and Sr25 dataset from online sources[4], EXP dataset from (Abboud et al., 2020), Random graph dataset from (Chen et al., 2020), 2D-Grid and Band-Pass dataset from (Balcilar et al., 2021), Zinc12K from (Dwivedi et al., 2020), Mnist-75 dataset from online source[5] which was used in (Balcilar et al., 2021) with exactly the same procedure, PROTEINS, ENZYMES, MUTAG and PTC from TU dataset (Morris et al., 2020) downloaded from resources of (Xu et al., 2019). All dataset except for EXP, Random and 2-D grid graph were used on a single task. We used EXP for graph isomorphism test and binary classification task. 2D-Grid graph was used for three different node regression tasks respectively on low-pass, band-pass and high-pass filtering effect prediction. Finally, Random graph is used on five different substructure counting tasks.

In all cases, we used roughly 30K trainable parameters for all problems and all models. We tuned the number of layers from 2 to 5 and the number of convolution kernels in Chebnet from 3 to 5. We used Adam optimization with learning rate in $[10^{-2}, 10^{-3}]$ and a weight decay in $[10^{-3}, 10^{-4}, 10^{-5}]$. We also used dropout layer before all graph convolution layers under selection of $[0, 0.1, 0.2]$ dropout rate. We used ReLU as non-linearity operation in all layers if it is not mentioned explicitly for any specific model. For classification problems, the loss function was implemented through cross-entropy. For regression problems, mean squared error was used as the loss function except on Zinc12K dataset where the loss function was mean absolute error. Unless otherwise specified, we used both sum and max readout layer after last layer of graph convolution. It is then followed by a fully connected layer which ended up with output layer.

In GNNML3, we use the eigendecomposition of normalized Laplacian to calculate the initial edge feature for all problems, except for Zinc12K and substructure counting problems where the eigen decomposition was performed on the adjacency. Each initial convolution support is set such

[4] http://users.cecs.anu.edu.au/∼bdm/data/graphs.html

[5] https://graphics.cs.tu-dortmund.de/fileadmin/ls7-www/misc/cvpr/mnist-superpixels.tar.gz

Table 5. Results on TU datasets. The values are the accuracy. Edge features are not used even if they are available in the datasets. The models use a one-hot encoding of node labels as node features, while the models also use extra 18 length continuous node features for ENZYMES-cont.

MODEL	MUTAG	ENZYMES	ENZYMES-CONT	PROTEINS	PTC
MLP	86.6% ± 4.95	30.8% ± 4.26	70.6% ± 5.22	74.3% ± 4.88	62.9% ± 5.89
GCN	89.1% ± 5.81	49.0% ± 4.25	74.2% ± 3.26	75.2% ± 5.11	64.3% ± 8.35
GAT	90.1% ± 5.84	54.1% ± 5.15	73.7% ± 4.47	75.9% ± 4.26	65.7% ± 7.97
GIN	89.4% ± 5.60	55.8% ± 5.23	73.3% ± 4.48	76.1% ± 3.97	64.6% ± 7.00
CHEBNET	89.7% ± 6.41	63.8% ± 7.92	75.3% ± 4.63	76.4% ± 5.34	65.5% ± 4.94
PPGN	90.2% ± 6.62	55.2% ± 5.44	72.9% ± 4.18	77.2% ± 4.53	66.2% ± 6.54
GNNML1	90.0% ± 0.42	54.9% ± 5.97	76.9% ± 5.14	75.8% ± 4.93	63.9% ± 6.37
GNNML3	90.9% ± 5.46	63.6% ± 6.52	78.1% ± 5.05	76.4% ± 5.10	66.7% ± 6.49

Table 6. Summary of the datasets used in our experiments.

	GRAPH8C	SR25	EXP	2D-GRID	RANDOM	BAND-PASS	PROTEINS	ENZYMES	MUTAG	PTC	MNIST-75	ZINC12K
TASK	ISO	ISO	ISO&2CLASS	NREG	REG	2CLASS	2CLASS	6CLASS	2CLASS	2CLASS	10CLASS	REG
GRAPHS	11117	15	1200	3	5K	5K	1113	600	188	344	70K	12K
NODES	8.0	25.0	44.44	900.0	18.8	200.0	39.06	32.63	17.93	25.55	75.0	23.15
EDGES	28.82	300.0	110.21	3480.0	62.67	1072.6	72.82	62.14	39.58	51.92	694.7	49.83
FEATURE	MONO	MONO	MONO	1	MONO	1	3LABEL	3LABEL+18	7LABEL	19LABEL	1	21LABEL
TRAIN	NA	NA	800	1	1500	3K	9-FOLD	9-FOLD	9-FOLD	9-FOLD	55K	10K
VAL	NA	NA	200	1	1000	1K	1-FOLD	1-FOLD	1-FOLD	1-FOLD	5K	1K
TEST	NA	NA	200	1	2500	1K	NA	NA	NA	NA	10K	1K

that $\Phi_s(\lambda) = exp(-b(\lambda - f_s)^2)$, where the bandwidth parameter b is set to the value of 5. The spectrum has been uniformly sampled between minimum eigenvalue and the maximum eigenvalue with a selection of $s_n = [3, 5, 10]$ points in order to select the band specific parameter. Thus, band specific parameter of each frequency profile can be written $f_s = \lambda_{min} + \frac{s-1}{s_n-1}(\lambda_{max} - \lambda_{min})$ for $s \in \{1, \dots, s_n - 1\}$. For the convolution support $s = 0$, we used all-pass filtering named identity matrix whose frequency response is $\Phi_0(\lambda) = \mathbf{1}$. Thus, we have a total of s_n convolution supports. The 1-hop distance is always used for receptive field which corresponds to $M = A + I$. For the learning of convolution supports needed in Eq.(5), we used a single layered MLP in each mlp_k where $mlp_1, mlp_2, mlp_3 : \mathbb{R}^S \to \mathbb{R}^{2S}$ with a sigmoid activation, and $mlp_4 : \mathbb{R}^{4S} \to \mathbb{R}^S$ with ReLU activation as long as S is the number of initial convolutions extracted in the preprocessing step. In Eq.(6), the size of the output of mlp_5 and mlp_6 is another hyperparameter where we used the same length with the first part of the Eq.(6) defined by dimension of $W^{(l,s)}$.

Mentioned hyperparamters are optimzed for concerned model according to validation set performance if it is available. For TU dataset, since the validation and test set is not available in public split, we first created a hyperparameter tuning task by dividing the dataset one time into pre-training (80%) and pre-validation (20%). The optimal value of the parameters is searched on the basis of the performance on the pre-validation set. Then, these hyperparameter values for the general test procedure as defined in (Xu et al., 2019).

Our tests were conducted with implementations of Chebnet, GCN, GIN and GAT layer provided by pytorch-geometric (Fey & Lenssen, 2019). Besides, PPGN, GNNML1 and GNNML3 layer were implemented as a class of pytorch-geometric and our models were tested on the basis of these implementation. By doing so, we integrate the PPGN into the widely used graph library pytorch-geometric and make it publicly available beside our own proposals.

H. Summary of the Baseline Models

H.1. MPNN Baselines

In this section of the appendix, we present the baseline methods which are GCN, GIN, Chebnet and GAT thanks to the general framework given by Eq.(1). Each model differs from others by selection of their convolution support C.

GCN uses a single convolution support given by;

$$C = (D + I)^{-0.5}(A + I)(D + I)^{-0.5}, \qquad (14)$$

where D is the diagonal degree matrix (Kipf & Welling, 2017) in Eq.(1).

Chebnet relies on the approximation of a spectral graph analysis proposed in (Hammond et al., 2011), based on the Chebyshev polynomial expansion of the scaled graph Laplacian. The number of convolution supports $C^{(k)}$ can be chosen. They are defined by (Defferrard et al., 2016) as follows:

$$\begin{aligned} C^{(1)} = I, \;\; C^{(2)} = 2L/\lambda_{\max} - I, \\ C^{(k)} = 2C^{(2)}C^{(k-1)} - C^{(k-2)}, \quad \forall k \geq 2. \end{aligned} \qquad (15)$$

Graph Isomorphism Network (GIN) defined in (Xu et al.,

2019) has a single convolution support defined as follows:

$$C = A + (1 + \epsilon)I, \tag{16}$$

where ϵ is a parameter that makes the support trainable. Another version named GIN-0 is also defined in the same paper where $\epsilon = 0$, which makes $C = A + I$. GIN proposes to use a desired number of MLP after each graph convolution. In our implementation, we use one MLP ($C = I$) after each GIN graph convolution as described in (Xu et al., 2019).

Graph attention networks (GATs) in (Veličković et al., 2018) proposes to transpose the attention mechanism from (Vaswani et al., 2017) into the graph world by the way of sparse attention instead of full attention in transformers. GAT convolution support can be seen as weighted, self loop added adjacency. It can be represented in Eq.(1) by defining its trainable convolution supports as follows:

$$\left(C^{(l,s)}\right)_{v,u} = \frac{e_{v,u}}{\sum_{k \in \tilde{\mathcal{N}}(v)} e_{v,k}}, \tag{17}$$

where $e_{v,u} = \exp\left(\sigma(\mathbf{a}^{(l,s)}[H^{(l)}_{:v}W^{(l,s)} || H^{(l)}_{:u}W^{(l,s)}])\right)$, and $\mathbf{a}^{(l,s)}$ is another trainable weight. Convolution support will be calculated from node v to each element of $\tilde{\mathcal{N}}(v)$, which shows the self-connection added neighborhood. In application of GAT, we use concatenation instead of sum in Eq.(1) where the paper proposed both and there is slightly empirical advantage to use concatenation.

All MPNN baselines start with a given node features $H^{(0)}$ and provide the node representation of the next layer by Eq.(1). After the last layer, we apply a graph readout function which summarizes the learned node representation. Graph readout layer is followed by a desired number of fully connected layers ended with a number of neuron defined by targeted number of classes.

H.2. PPGN Baseline

PPGN (Maron et al., 2019a) starts the process with a 3-dimensional input tensor where the adjacency, edge features (if it exists) and diagonalized node features are stacked on the 3rd dimension as:

$$H^{(0)} = [A|E_1|\cdots|E_e|diag(X_1)|\cdots|diag(X_d)]. \tag{18}$$

Here, $X \in \mathbb{R}^{n \times d}$ gathers node features and X_i is its i-th column vector, $E \in \mathbb{R}^{n \times n \times e}$ is edge features and $E_i \in \mathbb{R}^{n \times n}$ is its i-th edge feature matrix, thus initial feature tensor is $H^{(0)} \in \mathbb{R}^{n \times n \times (1+e+d)}$.

One layer forward calculation of PPNN would be:

$$H^{(l+1)} = m_3\left(\left[m_1(H^{(l)}) \circ m_2(H^{(l)}) | H^{(l)}\right]\right) \tag{19}$$

where $m_1, m_2 : \mathbb{R}^{n \times n \times d_{inp}} \to \mathbb{R}^{n \times n \times d_{mid}}$ and $m_3 : \mathbb{R}^{n \times n \times d_{mid}+d_{inp}} \to \mathbb{R}^{n \times n \times d_{out}}$ are trainable models that can be implemented by a one layer MLP followed by nonlinearity. d_{inp} is the feature length on the 3rd dimension. d_{mid}, d_{out} are the feature lengths which can be seen as hyperparameters of the layer. Multiplication ($\circ$) operates between matching features and means 2d matrix multiplication for each slice which has $n \times n$ dimensions. $|$ operator is just the concatenation of two tensor on the 3rd dimension. The output of the model would be:

$$Y = \sum_{l=1} mlp_l\left(\sum \text{diag}(H^{(l)}) \mid \sum \text{offdiag}(H^{(l)})\right). \tag{20}$$

We assign a function which selects the diagonal of each 2d slices of tensor as diag : $\mathbb{R}^{n \times n \times d} \to \mathbb{R}^{n \times 1 \times d}$ and function for selection the element out of the diagonal as offdiag : $\mathbb{R}^{n \times n \times d} \to \mathbb{R}^{n \times (n-1) \times d}$. We use the sum operator which performs sum over the first 2 dimensions as $\sum : \mathbb{R}^{d_1 \times d_2 \times d} \to \mathbb{R}^d$ and a trainable model that may be implemented by an MLP $mlp_l : \mathbb{R}^{2d} \to \mathbb{R}^{d_y}$, transforms the given vector into the targeted output representation length.

The one can see that in each layer, PPNN keeps $H^{(l)} \in \mathbb{R}^{n \times n \times d_l}$, thus its memory usage is in $\mathcal{O}(n^2)$. Since there is a matrix multiplication in Eq.(19), its computation complexity is in $\mathcal{O}(n^3)$ when using the naive matrix multiplication operations. The PPNN paper mentioned that the computational complexity can be decreased by using effective matrix multiplication, but it is the same for all algorithms as well. For this reason, we think that taking the naive implementation into account makes more sense to do a fair comparison. In addition, again because of matrix multiplication, its update mechanism is not local. Because of calculation of the u, v node pairs representation in Eq.(19), it needs to perform $\sum_k H^{(l)}_{u,k}.H^{(l)}_{k,v}$. That means that for each pair of nodes, k should be all nodes in the graph regardless how far away the node k from the concerned nodes u, v. In other words, very far away nodes feature affect the concerned node.

Even though PPNN (Maron et al., 2019a) is a very straight forward algorithm and has provable 3-WL power, the experimental results reported in the papers are not at the state of the art (Maron et al., 2019a; Dwivedi et al., 2020). We believe that this can be at least partly explained by some implementation problems. Indeed, it was implemented by gathering same size graphs into batches in order to handle graphs of different size in a dataset. So the batches do not consist of randomly selected graphs in each epoch during the training phase. In our implementation, we first find the maximum size of the graph denoted as n_{max}. Then, we create an initial tensor in Eq.(18) in dimension of $\mathbb{R}^{n_{max} \times n_{max} \times 1+e+d}$ where left top $n \times n \times 1 + e + d$ part of the tensor is valid, and the rest is zero. We also keep the valid part of the tensor diagonal and out of diagonal part mask in $M_0, M_1 \in \{0, 1\}^{n_{max} \times n_{max}}$ that shows which element is valid in the diagonal and which element

is valid out of the diagonal of the representation tensor. Since some part of the tensor $H^{(l)}$ are not valid, we need to prevent to assign value after application of trainable model m_k in Eq.(19), because it affects the matrix multiplication result. One solution may be to mask the MLP result by $M_0 + M_1$. Finally, we implement Eq.(20) by selection diagonal and off-diagonal element by previously prepared mask matrices by $\sum \text{diag}(H^{(l)}) = \sum M_0 \odot H^{(l)}$ and $\sum \text{offdiag}(H^{(l)}) = \sum M_1 \odot H^{(l)}$. By doing so, we can put any graph into same batch. These principles have been implemented as a class of the widely used open-source pytorch geometric library.