Repetition Makes Perfect: Recurrent Graph Neural Networks Match Message Passing Limit

Eran Rosenbluth, Martin Grohe
RWTH Aachen University
rosenbluth@informatik.rwth-aachen.de, grohe@informatik.rwth-aachen.de

Abstract

We precisely characterize the expressivity of computable Recurrent Graph Neural Networks (recurrent GNNs). We prove that recurrent GNNs with finite-precision parameters, sum aggregation, and ReLU activation, can compute any graph algorithm that respects the natural message-passing invariance induced by the Color Refinement (or Weisfeiler-Leman) algorithm. While it is well known that the expressive power of GNNs is limited by this invariance [Morris et al., AAAI 2019; Xu et al., ICLR 2019], we establish that recurrent GNNs can actually match this limit. This is in contrast to non-recurrent GNNs, which have the power of Weisfeiler-Leman only in a very weak, "non-uniform", sense where each graph size requires a different GNN to compute with. Our construction introduces only a polynomial overhead in both time and space.

Furthermore, we show that by incorporating random initialization, for connected graphs recurrent GNNs can express all graph algorithms. In particular, any polynomial-time graph algorithm can be emulated on connected graphs in polynomial time by a recurrent GNN with random initialization.

1 Introduction

Graph Neural Networks Message-Passing Graph Neural Networks (GNNs) (Kipf and Welling 2017; Gilmer et al. 2017) is a class of graph-processing architectures commonly used in tasks of learning on graphs. As such, characterizing their expressivity is of great importance.

A GNN is a finite sequence of operations, called *layers*, applied in parallel and in sync to each node. Its inputs are graphs whose nodes are assigned an initial feature-vector of certain dimension. In the broadest theoretical setting the vector is over the real numbers, however, most expressivity studies consider more restrictive domains: Some consider integers, some consider a compact segment $[a, b] \subset \mathbb{R}$, and most consider a finite domain. A layer starts with constructing a *message* from each neighbor, often being simply the neighbor's current value, although more sophisticated algorithms can also be used. Importantly, messages bear no identification of the sending node. An aggregation algorithm, typically dimension-wise sum; avg; or max, is then applied to the multiset of messages, producing a fixed-dimension value. Finally, a combination algorithm in the form of a *Multilayer Perceptron* (MLP) is applied to the aggregation value and the node's current value, producing the node's new value. For node-level tasks, a node's value after the application of the last layer is considered the output of the GNN for that node. For graph embeddings, the nodes' final values are aggregated, and an MLP is applied to the aggregation value, producing the GNN's output for the graph.

The node-centric, node-indifferent, nature of the algorithm means every GNN can technically be applied to graphs of all sizes and GNNs are isomorphism-invariant. The MLP part of the layers gives GNNs their learnability qualities.

It is common for GNNs to use the same aggregation type in all layers. Those that use sum are denoted *Sum-GNNs*.

The expressivity boundaries of GNNs can be attributed to the combination of three factors:

1. The message-passing scheme. A main consequence of this is a distinguishing power that cannot exceed that of the Color Refinement algorithm - or 1-dimensional *Weisfeiler-Leman* (1-WL) as it is sometimes referred to (Xu et al. 2019; Morris et al. 2019; Aamand et al. 2022).
2. The fixed number of layers-executions. An obvious effect of that is a fixed information-radius for each node's computation. Another effect is impossibility to enhance the expressivity of the layers' algorithms (see next) by means of repetition.
3. The combination and aggregation functions that make GNNs' layers. For reasons of learnability and runtime performance, these are of specific classes as mentioned above. While MLPs are universal approximators of continuous functions on compact domains (Hornik, Stinchcombe, and White 1989), their expressivity is very limited for non-compact domains e.g. no MLP can approximate $\forall x \in \mathbb{N}\ x \mapsto x^2$. As for the aggregation functions, all mentioned choices potentially lose information about the multiset of neighbors' messages.

Expressivity Notions In more than a few studies e.g. (Chen et al. 2019; Xu et al. 2019), an architecture is said to express a function f if and only if it *non-uniformly* expresses f: *For every graph size n there exists a model M_n of that architecture, that approximates f on all graphs* ***of that size***. Non-uniform expressivity is not only a weak guarantee in theory, but it also has limited relevance to practice: First, non-uniform expressivity, with M_n having a non-

polynomial size complexity, implies models that are too large already for small graphs. Second, even when the size-complexity is polynomial e.g. (Grohe 2023), non-uniform GNN models may succeed at inference time only when the input-graph size does not exceed the sizes of graphs in training time, otherwise they fail miserably - as implied by the theory and observed in experiments (Rosenbluth, Toenshoff, and Grohe 2023; Rosenbluth et al. 2024).

The strongest and more meaningful notion of expressivity, both to theory and practice, and the one that we use in this paper, is *uniform* expressivity. Here, an architecture is said to express a function f if and only if it *uniformly* expresses f: *There exists a model M of that architecture, that approximates f* ***on graphs of all sizes***. In informal terms, a uniform model computes an algorithm - accepting any input size, while a non-uniform one computes a sort of lookup table for the finitely many graphs of a specific size. From a practical standpoint, uniform expressivity means it may be possible to train a GNN model on graphs of smaller sizes and this model will approximate the target function also on graphs of larger sizes. We are not aware of any meaningful tight bounds shown for the uniform expressivity of GNNs. Obviously, GNNs cannot uniformly express functions that depend on an unbounded information-radius, but their limitations go beyond that: Previous works have shown that common GNN architectures cannot uniformly express basic regression and classification functions even when they are expressible by another GNN architecture and the required information-radius is only 2 (Rosenbluth, Toenshoff, and Grohe 2023; Grohe and Rosenbluth 2024).

Recurrent Graph Neural Networks Recurrent GNNs (Scarselli et al. 2008; Gallicchio and Micheli 2010) are similar to GNNs in that a recurrent GNN consists of a finite sequence of layers and those comprise the same algorithms of (non-recurrent) GNN layers. However, in a recurrent GNN the sequence can be reiterated a number of times that depends on the input. Recurrent GNNs have been used successfully in practice (Li et al. 2016; Selsam et al. 2016; Bresson and Laurent 2018; Tönshoff et al. 2023) and are considered promising architectures for solving various learning tasks. In terms of their uniform expressivity, recurrent GNNs are still limited by factor (1) as they still consist of local algorithms; by definition factor (2) does not apply to them; and the question is to what extent recurrence can mitigate factor (3). Recurrent MLPs with ReLU activation are proved to be Turing-complete (Siegelmann and Sontag 1992), implying that a layer's expressivity may be increased by reiterating it. However, it has not been clear thus far if reiteration can recover the information lost in aggregation.

A recurrent message-passing aggregate-combine architecture, where the nodes are aware of the graph size, was shown to be as expressive as message-passing can be (Pfluger, Cucala, and Kostylev 2024). However, the layers' functions there are not restricted to be computable, let alone an MLP and a common aggregation function, making the result only an upper bound for computable recurrent GNNs. For recurrent sum-aggregation GNNs, it has been proven that a single layer GNN can distinguish any two graphs distinguishable by Color Refinement (Bravo, Kozachinskiy, and Rojas 2024). However, the proof is existential, the activation function of the GNN must not be ReLU, and most importantly, the GNN has a parameter whose value must be a real number - of infinite precision - making it incomputable. A tight logical bound for a computable recurrent GNN architecture, named by the authors an 'R-Simple AC-GNN[F]', is proven in (Ahvonen et al. 2024). The architecture is defined to operate only with fixed-length float values, making it limited in one aspect, and to aggregate multisets of such values always in order (e.g. ascending), making it sophisticated in another. All in all, expressivity-wise it is a strictly weaker architecture, and essentially different, than the recurrent GNNs we study. Moreover, it is characterized in different terms than the ones we use to tightly characterize our architectures.

Message-Passing Limit By their definition, the uniform expressivity of computable recurrent GNNs is upper-bounded by the expressivity of a general message-passing algorithm with id-less nodes that are aware of the graph-size: A single recurring algorithm operating in parallel at each node, whose input at each recurrence is the node's value and a multiset of messages from the node's neighbors, and its output is a new value and a message to send. Importantly, the same message is broadcasted to all neighbors, and messages are received with no identification of the sending node. Recurrent GNNs are then a specific case, where the recurring algorithm is an MLP composed on an aggregation of the messages, rather than a general algorithm operating on the multiset of messages.

A well-studied representation of a node defines another important upper-bound: The class of all algorithms $\mathcal{A}(G, v)$, for a graph and node, that are invariant to the Color Refinement (CR) (or 1-Weisfeiler-leman) representation of the node. The Color Refinement procedure goes back to (Morgan 1965; Weisfeiler and Leman 1968), see also (Grohe 2021) and Section 3.

New Results We prove meaningful tight bounds for the uniform expressivity of computable recurrent GNNs - with finite precision weights and ReLU activation. All the reductions are achieved with polynomial time and space overhead. We assume a finite input-feature domain, and that the original feature is augmented with the graph-size value. There, we show that a recurrent single-layer sum-aggregation GNN can compute the following (see Figure 1 for an illustration):

1. Node-level functions:
   a. Any algorithm that is invariant to the Color Refinement (CR) value of the node. (Thm 4.2)
   b. Any message-passing algorithm. (Implied by (a))
   c. When adding global aggregation, any algorithm that is invariant to the Weisfeiler-Leman (WL) value of the node. (Thm. 5.3)
   d. For connected graphs, with random node initialization, any algorithm that is isomorphism-invariant (Cor. 5.2).
2. For connected graphs, any computable graph embedding that is invariant to the CR value of the graph. (Thm. 5.1)

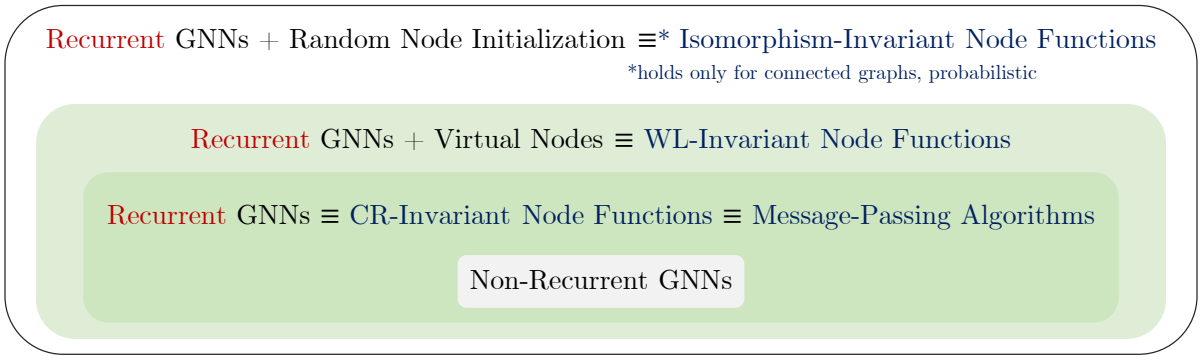

Figure 1: Uniform expressivity hierarchy. 'CR' and 'WL' are acronyms for Color Refinement and Weisfeiler-Leman. The results in this paper are the equivalencies. 'Node Functions' means computable functions $f(G, v)$ operating on a graph and node.

Roadmap In Section 3 we describe the color of a node and its relation to the message-passing scheme, and use it to define our upper-bound function classes. In Section 4 we define our recurrent GNN architecture and describe the reduction from the upper-bound classes through intermediate models and down to our architecture. In Section 5 we extend our results to graph embeddings and to recurrent GNN architectures that go beyond pure message-passing.

2 Preliminaries

By $\mathbb{N}; \mathbb{Q}; \mathbb{R}$ we denote the natural;rational; and real numbers respectively. We define $\mathbb{Q}_{[0,1]} \coloneqq \{q : q \in \mathbb{Q}, 0 \leq q \leq 1\}$. Let $v \in \mathbb{R}^d$ be a d-dimension vector, we denote by $v(i)$ the value at position i, and by $v[a, b] \coloneqq (v_a, \ldots, v_b)$ the sub-vector from position a to b. Let $v \in \mathbb{R}^d$ and $a \in \mathbb{R}$, we define $v + a \coloneqq v(1) + a, \ldots, v(d) + a$. Let $v_1, v_2 \in \mathbb{R}^d$ be d-dimension vectors, we define $v_1 + v_2 \coloneqq v_1(1) + v_2(1), \ldots, v_1(d) + v_2(d)$. For vectors $u \in \mathbb{R}^m, v \in \mathbb{R}^n$ we define $u, v \coloneqq u(1), \ldots, u(m), v(1), \ldots, v(n)$.
For a set S, we denote the set of all finite multisets with elements from S by $\left(\!\binom{S}{*}\!\right)$. We denote the set of all finite tuples with elements from S by S^*. For a vector $v \in \mathbb{R}^d$ we define $\dim(v) \coloneqq d$, and for a matrix $W \in \mathbb{R}^{d_1 \times d_2}$ we define $\dim(W) \coloneqq (d_1, d_2)$.

We define $\mathcal{B} \coloneqq \{0, 1\}^*$ the set of all finite binary strings, and $\mathcal{B}_k \coloneqq \{0, 1\}^k$ the set of all binary strings of length k. For $x \in \mathcal{B}_k$ we define $|x| \coloneqq k$.

A (vertex) *featured graph* $G = \langle V(G), E(G), S, Z(G) \rangle$ is a 4-tuple being a graph with a *feature map* $Z(G) : V(G) \to S$, mapping each vertex to a value in some set S. Let $v \in V(G)$, we denote $Z(G)(v)$ also by $Z(G, v)$. We define the *order*, or *size*, of G, $|G| \coloneqq |V(G)|$. A *rooted graph* is a pair (G, v) where G is a graph and $v \in V(G)$ is a vertex. We denote the set of graphs featured over S by $\mathcal{G}_S$ and the set of all featured graphs by $\mathcal{G}_*$. Note that in this paper we focus on the graph domain $\mathcal{G}_\mathcal{B}$ i.e. single-dimension featured graphs where the feature is a bit-string. However, our results apply to multi-dimensional featured graphs as well: A node's tuple can be encoded into one dimension during pre-processing, or, alternatively, the construction in our proofs can be extended such that the initial stage of the computation includes encoding the tuple into one dimension.

We denote the set of all feature maps that map to some set T by $\mathcal{Z}_T$, and we denote the set of all feature maps by $\mathcal{Z}_*$. Let $\mathcal{G} \subseteq \mathcal{G}_*$, a mapping $f : \mathcal{G} \to \mathcal{Z}_*$ to new feature maps is called a *feature transformation*.

A *message-passing algorithm* [1] is a pair $C = (A, f)$, $A \in \mathcal{B}$, $f : \mathcal{B}^2 \to \mathcal{B}^2$ comprising an initial state and a computable function. It defines a feature transformation $C : \mathcal{G}_\mathcal{B} \to \mathcal{Z}_\mathcal{B}$ as follows: Define

$$\forall G \in \mathcal{G}_\mathcal{B}\ \forall v \in V(G)\ C^{(0)}(G, v) \coloneqq \theta\big(A, |G|, Z(G, v)\big), \emptyset$$

for some encoding θ of the initial state; the graph size; and the initial feature, and $\emptyset$ denoting the empty string. Then, define $\forall i > 0\ \ C^{(i+1)}(G, v) \coloneqq$

$$f\Big(C^{(i)}(G, v)(1), \mu(\{\{C^{(i)}(G, w)(2) : w \in N_G(v)\}\})\Big)$$

for some multiset encoding μ. Define the first iteration when a 'finished' indicator turns 1:

$$I_v \coloneqq \begin{cases} \min(i : C^{(i)}(G, v)(1)(1) = 1) & \text{min exists} \\ \infty & \text{otherwise} \end{cases}$$

Finally, $C(G, v) \coloneqq C^{(I_v)}(G, v)(2)$, if I_v is defined.

Throughout the paper we refer to Multilayer Perceptrons (MLPs), meaning specifically ReLU-activated MLPs, formally defined in (Rosenbluth and Grohe 2025, Appendix). A d dimension *recurrent MLP* F is an MLP of I/O dimensions $d; d$. It defines an iterative function $f^{(t)}, t \in \mathbb{N}$ such that $f_F^{(0)}(x) \coloneqq x,\ \ \forall t > 0\ f_F^{(t)}(x) \coloneqq f_F\big(f_F^{(t-1)}(x)\big)$.

3 Message-Passing Information

In this section, we give a precise technical description of the limits of message-passing algorithms using the *Color Refinement* procedure. It aims to describe the maximal "message-passing information" each node can obtain. It will be convenient to refer to this message-passing information

[1] Our definition is equivalent to any distributed algorithm where the initial input includes the graph size, and the input from a node's neighbors is a multiset - with no ids or order

as the "color" of a node. Note, however, that the message-passing colors are complex objects, nested tuples of multi-sets, which will later be compactly represented by a directed acyclic graph (dag).

Definition 3.1. *Let $G \in \mathcal{G}_{\mathcal{B}_k}$. For every $t \geq 0$ and $v \in V(G)$, we define the* message-passing color of v after t rounds *inductively as follows: The* initial color *of v is just its feature in G, that is, $\mathsf{mpc}_G^{(0)}(v) \coloneqq Z(G,v)$. The color of v after $(t+1)$ rounds is the color of v after t rounds together with the multiset of colours of v's neighbours, that is,*

$$\mathsf{mpc}_G^{(t+1)}(v) \coloneqq \big(\mathsf{mpc}_G^{(t)}(v), \{\!\{\mathsf{mpc}_G^{(t)}(w) \mid w \in N_G(v)\}\!\}\big).$$

Moreover, we define the final color *of v to be*

$$\mathsf{mpc}_G(v) \coloneqq \mathsf{mpc}_G^{2|G|}(v).$$

For all $t, n, k \in \mathbb{N}$, we let

$$\mathsf{MPC}_{n,k}^{(t)} \coloneqq \big\{\mathsf{mpc}_G^{(t)}(v) \;\big|\; G \in \mathcal{G}_{\mathcal{B}_k}, |G| = n, v \in V(G)\big\},$$
$$\mathsf{MPC}^{(t)} \coloneqq \textstyle\bigcup_{n,k\in\mathbb{N}} \mathsf{MPC}_{n,k}^{(t)}, \qquad \mathsf{MPC} \coloneqq \textstyle\bigcup_{t\in\mathbb{N}} \mathsf{MPC}^{(t)}.$$

While most applications of Color Refinement are mainly interested in the partition of the vertices into color classes, for us, the actual colors carrying the message-passing information are important. If written as strings in a straightforward manner, the colors will become exponentially large (up to size $\Omega(n^t)$). We may also view the colors by trees, which are still exponentially large. We introduce a polynomial-size dag representation for each color $c \in \mathsf{MPC}_{n,k}^{(2n)}$, denoted $D(c)$. For the full description of the dag construction, please refer to (Rosenbluth and Grohe 2025, Appendix).

A feature transformation $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ is *message-passing-invariant* (for short: *mp-invariant*) if for all graphs $G, H \in \mathcal{G}_{\mathcal{B}}$ of the same order $|G| = |H|$ and nodes $v \in V(G), w \in V(H)$, if $\mathsf{mpc}_G^{(t)}(v) = \mathsf{mpc}_H^{(t)}(w)$ for all $t \geq 1$ then $F(G,v) = F(H,w)$. By induction on the number of message-passing rounds, every feature transformation computed by a message-passing algorithm is mp-invariant. The converse is implied by the fact that, clearly, $D(\mathsf{mpc}_G(v))$ can be constructed by a message-passing algorithm, together with the following lemma which asserts that, remarkably, $D(\mathsf{mpc}_G(v))$ suffices to compute any mp-invariant feature transformation.

Lemma 3.2. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be a computable feature transformation. Then F is mp-invariant if and only if there is an algorithm that computes $F(G,v)$ from $\mathsf{mpc}_G(v)$. More precisely, there is an algorithm that, given $D(\mathsf{mpc}_G(v))$, computes $F(G,v)$, for all $G \in \mathcal{G}_{\mathcal{B}}$ and $v \in V(G)$. Furthermore, if F is computable in time $T(n)$ then the algorithm can be constructed to run in time $T(n) + \mathrm{poly}(n)$, and conversely, if the algorithm runs in time $T(n)$ then F is computable in time $T(n) + \mathrm{poly}(n)$.*

The crucial step towards proving this lemma is to reconstruct a graph from the message-passing color: Given $D(\mathsf{mpc}_G(v))$, we can compute, in polynomial time, a graph G' and a node v' such that $\mathsf{mpc}_{G'}(v') = \mathsf{mpc}_G(v)$. This nontrivial result is a variant of a theorem due to (Otto 1997). Once we have this reconstruction, Lemma 3.2 follows easily.

4 Main Result

We would like to characterize the expressivity of R-GNNs, which operate on rational numbers, in terms of computable functions i.e. algorithms that operate on bit-strings. We use two encodings of the latter representation by the former: Rational Quaternary and Rational Binary, the reasons for which reside in the proof of Lemma 4.8. Let

$$\mathrm{RQ} \coloneqq \{\Sigma_{i=1}^k a_i 4^{-i} : \forall j \in [k]\ a_j \in \{1,3\}, k \in \mathbb{N}\}$$

$$\mathrm{RB} \coloneqq \{\Sigma_{i=1}^k a_i 2^{-i} : \forall j \in [k]\ a_j \in \{0,1\}, k \in \mathbb{N}\}$$

, then define the encoding operations

$$\mathfrak{rq} : \mathcal{B} \to \mathrm{RQ},\ \mathfrak{rq}(b_1,\ldots,b_k) \coloneqq \Sigma_{i=1}^k (2b_i+1)4^{-i}$$

$$\mathfrak{rb} : \mathcal{B} \to \mathrm{RB},\ \mathfrak{rb}(b_1,\ldots,b_k) \coloneqq \Sigma_{i=1}^k b_i 2^{-i}$$

For vectors of binary strings we may use the $\mathfrak{rq}, \mathfrak{rb}$ notations to denote the element-wise encoding, that is, $\forall (B_1,\ldots,B_l) \in \mathcal{B}^l$

$$\mathfrak{rq}(B_1,\ldots,B_l) \coloneqq (\mathfrak{rq}(B_1),\ldots,\mathfrak{rq}(B_l)),$$

$$\mathfrak{rb}(B_1,\ldots,B_l) \coloneqq (\mathfrak{rb}(B_1),\ldots,\mathfrak{rb}(B_l))$$

We are now ready to define the recurrent GNN architecture that is the subject of our main result. Part of the definition is the initial input provided to it. We choose to include the maximum feature length (across all nodes) k in that input, as this allows us later to construct a single GNN for all $G \in \mathcal{G}_{\mathcal{B}}$, as stated in Theorem 4.2. Alternatively, we could waive having k in the input, make it instead a parameter of the architecture, and restrict the statement in Theorem 4.2 to all $G \in \mathcal{G}_{\mathcal{B}_k}$.

Definition 4.1. *A* Recurrent Sum-GNN *(R-GNN) $N = (A, F)$ of dimension d is a pair comprising a constant initial-state vector $A \in \mathbb{Q}^{d-3}$ and an MLP F of I/O dimensions $2d; d$. Dimension d is a 'computation finished' indicator, and dimension $d-1$ holds the computation result. It defines a feature transformation $N : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Let $G \in \mathcal{G}_{\mathcal{B}}$, let $k \coloneqq \max(|b| : b \in img(Z(G)))$ the maximum length over the binary-string features of the vertices in G, and let $v \in V(G)$. Define the initial value of N to be the rational vector comprising the graph size; max feature length; initial feature; and initial state, that is,*

$$N^{(0)}(G,v) \coloneqq |G|, k, \mathfrak{rb}(Z(G,v)), A$$

Define the value of N after $t > 0$ iterations to be

$$N^{(t)}(G,v) \coloneqq F(N^{(t-1)}(G,v), \Sigma_{w\in N_G(v)} N^{(t-1)}(G,w))$$

Define the first iteration when 'finished' turns 1

$$I_v \coloneqq \begin{cases} \min(i : N^{(i)}(G,v)(d) = 1) & \textit{min exists} \\ \infty & \textit{otherwise} \end{cases}$$

Then,

$$N(G,v) \coloneqq \begin{cases} \mathfrak{rb}^{-1}\big(N^{(I_v)}(G,v)(d-1)\big) & I_v \in \mathbb{N} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

the binary string represented in rational-binary encoding at position $(d-1)$, when the 'finished' indicator turns 1.

We define a time measure $T_N(G, v) \coloneqq I_v$*, and*

$$T_N(G) \coloneqq \max(I_v : v \in V(G))$$

We say that an R-GNN N uses time $T(n)$*, for a function* $T : \mathbb{N} \to \mathbb{N}$*, if for all graphs* G *of order at most* n *it holds that* $T_N(G) \leq T(n)$*. We define* $L_N(G, v)$ *to be the largest bit-length over all parameters' and neurons' values of* F*, at any point of the computation for* v*, and we define*

$$L_N(G) \coloneqq \Sigma_{v \in V(G)} L_N(G, v)$$

We say that an R-GNN N uses space $S(n)$*, for a function* $S : \mathbb{N} \to \mathbb{N}$*, if for all graphs* G *of order at most* n *it holds that* $L_N(G) \leq S(n)$*.*

Note that reaching a fixed point is not required for our results, hence it is not part of R-GNNs termination definition. However, the R-GNN we construct in the proof of Lemma 4.8 does have that property i.e. $I_v \in \mathbb{N} \Rightarrow \forall t \geq I_v \; N^{(t)}(G, v)[d-1, d] = N^{(I_v)}(G, v)[d-1, d]$, which may be useful in practice and for relation to logic.

In (Rosenbluth and Grohe 2025, Theorem B.2) it is proven that having the graph size in the input is a must for maximum expressivity. For recurrent GNNs with a mechanism known as global readout (see Section 5), the requirement is removed since such GNNs can compute the size.

Our main theorem refers to mp-invariant functions. However, since every message-passing algorithm is mp-invariant and since R-GNNs are specific message-passing algorithms, we have that R-GNNs are expressivity-wise equivalent also to message-passing algorithms.

Theorem 4.2. *Let* $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ *be a computable feature transformation. Then* F *is mp-invariant if and only if there is an R-GNN* N *such that*

$$\forall G \in \mathcal{G}_{\mathcal{B}} \; \forall v \in V(G) N(G, v) = F(G)(v)$$

Furthermore, if F *is computable in time* $T(n)$ *and space* $S(n)$ *then* N *uses time* $O(T(n)) + \mathrm{poly}(n)$ *and space* $O(S(n)) + \mathrm{poly}(n)$*.*

Intermediate Reductions

Our proof of Theorem 4.2 reduces an mp-invariant function to an R-GNN through a sequence of three intermediate computation models, see Figure 2 for a detailed illustration. The models operate on bit-strings, hence we define bit-encodings for data entities that appear in the models' definitions.

We define $\delta : \mathsf{MPC} \to \mathcal{B}$ to be an encoding of space complexity $O(n^3 \log n + kn)$ for all $c \in \mathsf{MPC}^{(t)}_{n,k}, t \leq 2n$, such that all required operations can be done in polynomial time. Following the construction description of $D(\mathsf{mpc}_G(v))$ in (Rosenbluth and Grohe 2025, Appendix), it is evident that there exists such an encoding.

We define $\mu : \left(\!\!\binom{\mathcal{B}}{*}\!\!\right) \to \mathcal{B}$ to be a multiset encoding such that the elements can be encoded and decoded in linear time by a Random Access Machine (RAM). We define $\theta : \mathcal{B}^* \to \mathcal{B}$ to be a tuple encoding such that the elements can be encoded and decoded in linear time by a RAM. Clearly, such encodings exist.

For $l, c, k \in \mathbb{N}$ we define $\theta^{(l)}_{k,c} : (\mathcal{B}_k)^l \to \mathcal{B}$ to be a tuple encoding of l bit-strings of length at most k, of space complexity $O(l(\log(c) + k)$, such that the elements can be encoded and decoded in linear time by a RAM, and, most importantly, the separation between the elements is preserved under summation of c such encodings: For all $(x_1, \ldots, x_c), x_i \in (\mathcal{B}_k)^l$ it holds that

$$\Sigma^c_{i=1}(\theta^{(l)}_{k,c}(x_i)) = \theta^{(l)}_{k,c}\Big(\Sigma^c_{i=1}\big(x_i(1)\big), \ldots, \Sigma^c_{i=1}\big(x_i(l)\big)\Big)$$

A straightforward encoding that reserves $\log(c2^k)$ bits for each one of the l parts satisfies the requirements.

Definition 4.3. *An* MPC Graph Algorithm *(MPC-GA)* $C = (M)$ *is simply a Turing machine. It defines a feature transformation* $C : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ *as follows:*
Let $G \in \mathcal{G}_{\mathcal{B}}, v \in V(G)$*, then* $C(G, v) \coloneqq M(\delta(\mathsf{mpc}_G(v)))$*.*

Let F be an mp-invariant function. By Lemma 3.2, there exists an MPC-GA that computes F with polynomial time and space overhead. Hence, the first stage in the reduction sequence in Figure 2 is already proven. The next step is to translate MPC-GA - whose input is already the color of a vertex - to a distributed algorithm that has to gather that information before applying the core algorithm to it.

Definition 4.4. *Let* $C = (A, f)$*,* $A \in \mathcal{B}$*,* $f : \mathcal{B}^2 \to \mathcal{B}^2$ *be a pair, comprising an initial state and a computable function. It defines a feature transformation* $C : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ *as follows: Let* $G \in \mathcal{G}_{\mathcal{B}}$ *and* $v \in V(G)$*, then define*

$$C^{(0)}(G, v) \coloneqq \theta\big(A, |G|, \delta(Z(G, v))\big), \delta\big(Z(G, v)\big)$$

a 2-dimension vector, the first binary string being the tuple encoding of the initial state; graph size; and encoding of the initial feature, and the second being only an encoding of the initial feature. Define $\forall t \geq 0 \;\; C^{(t+1)}(G, v) \coloneqq$

$$f\Big(C^{(t)}(G, v)(1), \mu(\{\!\{C^{(t)}(G, w)(2) : w \in N_G(v)\}\!\})\Big)$$

the value after $t + 1$ *iterations. Finally, define*

$$C(G, v) \coloneqq C^{(2|G|+1)}(G, v)(2)$$

That is, the final output is the second output of f *after the* $2|G| + 1$ *iteration. We say that* $C = (A, f)$ *is a* Message-Passing Limited Graph Algorithm *(MP-LGA) if*

$$\forall G \in \mathcal{G}_{\mathcal{B}_k} \; \forall \, v \in V(G) \; \forall t \in [2|G| + 1]$$

$$|\mu(\{\!\{C^{(t-1)}(G, w)(2) : w \in N_G(v)\}\!\})| \leq O(3k|G|^4)$$

, that is, the bit-length of the multiset of neighbors' messages does not exceed $3k|G|^4$*.*

Lemma 4.5. *Let* $C = (M)$ *be an MPC-GA, then there exist* $A \in \mathcal{B}$*,* $f : \mathcal{B}^2 \to \mathcal{B}^2$ *such that* $C' = (A, f)$ *is an MP-LGA and* $\forall G \in \mathcal{G}_{\mathcal{B}} \; \forall v \in V(G) \; C(G, v) = C'(G, v)$*. Furthermore,* C' *incurs polynomial time and space overhead.*

Next, we reduce the MP-LGA model to a model where the input from neighbors is **the sum** of the multiset of neighbors messages rather than the multiset itself. This addresses the first main obstacle of the overall reduction: To recover the information lost by the sum-aggregation and reconstruct the multiset of messages.

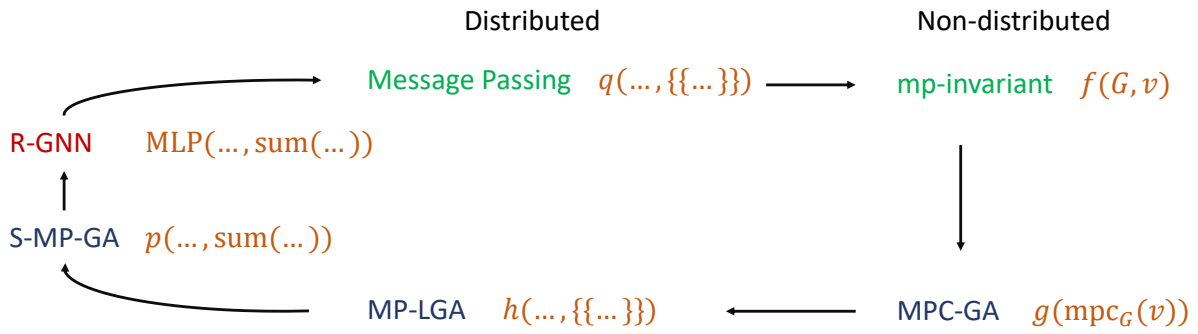

Figure 2: An overview of the reduction sequence from message-passing algorithms and mp-invariant functions to R-GNNs. Every message-passing algorithm is mp-invariant - by induction on the number of iterations. Then, starting from the mp-invariant class and moving clockwise, the reductions correspond to Lemma 3.2; Lemma 4.5; Lemma 4.7; and Lemma 4.8.

Definition 4.6. *Let $C = (A, f)$, $A \in \mathcal{B}$, $f : \mathcal{B}^4 \to \mathcal{B}^4$ be a pair, comprising an initial state and a computable function. It defines a feature transformation $C : \mathcal{G}_{\mathcal{B}_k} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Let $G \in \mathcal{G}_{\mathcal{B}_k}$, $v \in V(G)$, then define*

$$C^{(0)}(G,v) \coloneqq \theta\big(|G|, Z(G,v), A\big), \theta^{(2)}_{k,|G|}\big(1, Z(G,v)\big), 0, 0$$

a vector of 4 binary strings, the first being an encoding of the graph size; initial feature; and initial state, the second being an encoding of 1 and the initial feature, and the 3^{rd} and 4^{th} representing the final result and a 'finished' indicator. Define $\forall t \geq 0$ $C^{(t+1)}(G,v) \coloneqq$

$$f\Big(C^{(t)}(G,v)(1),\ \Sigma_{w \in N(v)} C^{(t)}(G,w)(2),\ C^{(t)}(G,v)[3,4]\Big)$$

the value after $t+1$ iterations. Define

$$I_v \coloneqq \begin{cases} \min(i : C^{(i)}(G,v)(4) = 1) & \textit{min exists} \\ \infty & \textit{otherwise} \end{cases}$$

$$C(G,v) \coloneqq \begin{cases} C^{(I_v)}(G,v)(3) & I_v \in \mathbb{N} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

That is, the result is the binary string at position 3, when the 'finished' indicator turns 1. We say that $C = (A, f)$ is a Sum MP Graph Algorithm *(S-MP-GA) if $\forall G \in \mathcal{G}_{\mathcal{B}_k}$ $\forall v \in V(G)$ $\forall t \in [2|G|+1]$ it holds that $|\Sigma_{w \in N(v)} C^{(t-1)}(G,w)(2)| \leq O(3k|G|^4)$ i.e. the bit-length of the sum of neighbors' messages is bounded by $3k|G|^4$.*

Besides having a sum aggregation, an S-MP-GA differs from an MP-LGA in two technical properties:

1. A message sent by a vertex consist of two parts, 'count' and 'value', rather than one. This is in order to count the number of sending vertices, which is useful later.
2. Two dimensions are used solely to define the final value, to make S-MP-GAs similar in that regard to R-GNNs.

The next lemma is a key step in our sequence of reductions.

Lemma 4.7. *Let $C = (A, f)$ be an MP-LGA then there exist A', f' such that $C' = (A', f')$ is an S-MP-GA and $\forall G \in \mathcal{G}_{\mathcal{B}}$ $\forall v \in V(G)$ $C'(G,v) = C(G,v)$. Furthermore, C' incurs polynomial time and space overhead.*

Reduction to R-GNN

Finally, we reduce S-MP-GAs to R-GNNs. The essential difference between the models is the recurring algorithm: In an S-MP-GA it can be any computable function i.e. a Turing machine, while in an R-GNN it is restricted to be an MLP.

Lemma 4.8. *Let $C = (B, h)$ be an S-MP-GA, then there exists an R-GNN $N = (A, F)$ such that $\forall G \in \mathcal{G}_{\mathcal{B}}$ $\forall v \in V(G)$ $C(G,v) = N(G,v)$. Furthermore, N incurs polynomial time and space overhead.*

Note that an R-GNN is, in a way, an extension of a recurrent MLP to the sum-aggregation message-passing setting. In (Siegelmann and Sontag 1992) it is shown that recurrent MLPs are Turing-complete. Let M be a Turing-machine that computes h, we would like to use the result in (Siegelmann and Sontag 1992) and emulate M using the recurrent MLP in an R-GNN. Yet, this requires overcoming two significant gaps:

1. An encoding gap. In (Siegelmann and Sontag 1992) the emulation of a Turing machine is done by emulating a two-stack machine where a stack's content is always represented as a value in RQ. Since RQ is not closed under summation, a naive attempt to use the sum of the neighbors' stacks directly - as input to the Turing machine emulation - is doomed to fail: The sum may be an invalid input and consequently the output will be wrong. To overcome this, we translate outgoing messages from RQ to RB - which is closed under summation, and we translate the incoming sum-of-messages back to RQ. The translations are implemented by two dedicated recurrent sub-MLPs of F.
2. A synchronization gap. In an S-MP-GA computation, nodes are synchronized by definition: The i^{th} recurrence's input is the the sum of results of the $i-1$ application of h to each of the neighbors. However, in an emulating R-GNN that is based on (Siegelmann and Sontag 1992), every recurrence corresponds merely to a Turing machine step. As different nodes may require a different number of steps to complete the computation of a single application of h, when a node finishes its i^{th} emulation

of h, in the t^{th} recurrence, its external input in recurrence $t+1$ is not necessarily the sum of its neighbors' i^{th} result, and it is unknown in what recurrence it will be. To overcome this, we augment the recurrent MLP described thus far with a recurrent sub-MLP of F that implements a synchronization algorithm across all nodes. That sub-MLP runs for a the same number of recurrences for all nodes , hence, in itself it is synchronized.

Overall, the recurrent MLP F consists of 8 recurrent sub-networks, each fulfilling a different task. See (Rosenbluth and Grohe 2025, Figure 5) for an outline of F's structure. At each iteration only one sub-network, other than the synchronizer, executes its task while the others execute a computation that does not affect their main outputs. See (Rosenbluth and Grohe 2025, Appendix) for further details.

5 Further Results

Our main theorem, Theorem 4.2, has a number of interesting variants and implications. First, it has a version for *graph embeddings* $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$. A graph-embedding R-GNN $N = (R, M)$ comprises an R-GNN R and an MLP M. It applies R, averages the nodes' values, and applies M. Formally, denote by I_v the last iteration of R on a node v, then

$$\forall G \in \mathcal{G}_{\mathcal{B}}\ N(G) \coloneqq \mathfrak{tb}^{-1}\Big(M\big(\tfrac{1}{|G|}\Sigma_{v\in V(G)}R^{(I_v)}(G,v)\big)\Big)$$

We say that two graphs are *mp-indistinguishable*, or indistinguishable by Color Refinement, if the same message-passing colors appear with the same multiplicities, that is, $\forall t \geq 0$

$$\{\{\mathsf{mpc}_G^{(t)}(v) \mid v \in V(G)\}\} = \{\{\mathsf{mpc}_H^{(t)}(v) \mid v \in V(H)\}\}$$

Note that this implies $|G| = |H|$. A function $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$ is *mp-invariant* if for all mp-indistinguishable graphs G, H we have $F(G) = F(H)$. Clearly, all graph embeddings computable by R-GNNs are mp-invariant. The following states the converse, which holds only for connected graphs, a limitation proved in (Rosenbluth and Grohe 2025, Thm. C.1).

Theorem 5.1. *Let $\mathcal{CG}_{\mathcal{B}} \subset \mathcal{G}_{\mathcal{B}}$ be the subset of connected graphs in $\mathcal{G}_{\mathcal{B}}$, and let $F : \mathcal{CG}_{\mathcal{B}} \to \mathcal{B}$ be computable. Then, F is mp-invariant if and only if there exists an R-GNN N such that $\forall G \in \mathcal{CG}_{\mathcal{B}}\ N(G) = F(G)$. Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then N uses time $O(T(n)) + \mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.*

So far, R-GNNs can only compute mp-invariant functions. For connected graphs, we can break the invariance by introducing *random node initialization* (RNI) (Abboud et al. 2021), that is, augmenting the initial feature of each node with a random number $r \sim U(0,1)$. GNNs with RNI describe a randomized algorithm, yet this randomized algorithm, or the random variable it computes, still satisfies the usual equivariance condition (see (Abboud et al. 2021)). We say that an R-GNN N with RNI computes a function $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ on a graph $G \in \mathcal{G}_{\mathcal{B}}$ and node $v \in V(G)$ if and only if: $\Pr(N(G,v) = F(G,v)) \geq p$ for some $\frac{1}{2} < p$, and $N(G,v) \neq F(G,v) \Leftrightarrow N(G,v) =$ null-value. By repeatedly running N we can boost $\Pr(N(G,v) = F(G,v))$ arbitrarily close to 1. See (Rosenbluth and Grohe 2025, Appendix) for further details.

Corollary 5.2. *Let $\mathcal{CG}_{\mathcal{B}} \subset \mathcal{G}_{\mathcal{B}}$ be the subset of connected graphs in $\mathcal{G}_{\mathcal{B}}$, and let $F : \mathcal{CG}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be computable in time $T(n)$ and space $S(n)$. Then, there exists an R-GNN N with RNI, such that F is computable by N. Furthermore, N uses time $O(T(n)) + \mathrm{poly}(n)$, space $O(S(n)) + \mathrm{poly}(n)$, and $O(n \log n)$ random bits.*

Another variant of our main result concerns a common extension of GNNs which is the addition of *global sum-aggregation* (a.k.a. *global sum*; *virtual nodes*), i.e. a sum-aggregation of the features of all nodes, as a third input to the combine MLP (Gilmer et al. 2017; Barceló et al. 2020). Adapting Definition 4.1, let $N = (A, F)$ be a GNN with global sum, then F has I/O dimensions $3d; d$, and

$$N^{(i+1)}(G,v) \coloneqq F\big(N^{(i)}(G,v), \mathrm{sum}_{w\in N_G(v)}N^{(i)}(G,w), \mathrm{sum}_{w\in V(G)}N^{(i)}(G,w)\big)$$

Note that here we do not need the size of the graph as an input, since it can easily be computed using global sum. Instead of mp-invariance, R-GNNs with global sum satisfy a different invariance that we call *WL-invariance*. It is based on a variant of the Color Refinement algorithm, the 1-dimensional Weisfeiler-Leman algorithm, that captures the additional global information obtained through global sum. See (Rosenbluth and Grohe 2025, Appendix).

Theorem 5.3. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be a computable feature transformation. Then F is WL-invariant if and only if there is an R-GNN with global aggregation that computes F. Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then the R-GNN uses time $O(T(n)) + \mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.*

This theorem also has a version for graph embeddings. Here it is not restricted to connected graphs, since global aggregation provides access to all connected components.

Corollary 5.4. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$ be computable. Then F is WL-invariant if and only if it is computable by an R-GNN.*

Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then the R-GNN uses time $O(T(n)) + \mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.

6 Concluding Remarks

We prove that recurrent graph neural networks can emulate any message-passing algorithm, with only a polynomial time and space overhead. Thus recurrent graph neural networks are universal for message-passing algorithms, or computable mp-invariant functions. Note that our theorem is not an approximation theorem; by focusing on computable functions, we can actually construct GNNs computing the functions exactly. By adding randomization, we can even overcome the limitation to mp-invariant functions.

Our time-complexity analysis for the reduction states "polynomial overhead", and it is not difficult to extract an upper bound of $O(n^{10}k^2)$ from our proofs. It will be useful to know whether our reduction can be improved so to have a lower complexity, and to have a lower bound for any reduction from computable mp-invariant functions to R-GNNs. Ideally, a tight bound would be constructively proven. These remain open.

Acknowledgments

During the work on this paper, Eran Rosenbluth was funded by the European Union (ERC, SymSim, 101054974) and German Research Council (DFG), RTG 2236 (UnRAVeL). Martin Grohe was funded by the European Union (ERC, SymSim, 101054974).

References

Aamand, A.; Chen, J.; Indyk, P.; Narayanan, S.; Rubinfeld, R.; Schiefer, N.; Silwal, S.; and Wagner, T. 2022. Exponentially improving the complexity of simulating the Weisfeiler-Lehman test with graph neural networks. *Advances in Neural Information Processing Systems*, 35: 27333–27346.

Abboud, R.; Ceylan, İ. İ.; Grohe, M.; and Lukasiewicz, T. 2021. The Surprising Power of Graph Neural Networks with Random Node Initialization. In Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, 2112–2118.

Ahvonen, V.; Heiman, D.; Kuusisto, A.; and Lutz, C. 2024. Logical characterizations of recurrent graph neural networks with reals and floats. *Advances in Neural Information Processing Systems*, 37: 104205–104249.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net.

Bravo, C.; Kozachinskiy, A.; and Rojas, C. 2024. On dimensionality of feature vectors in MPNNs. In *Forty-first International Conference on Machine Learning*.

Bresson, X.; and Laurent, T. 2018. Residual Gated Graph ConvNets.

Chen, Z.; Villar, S.; Chen, L.; and Bruna, J. 2019. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32.

Gallicchio, C.; and Micheli, A. 2010. Graph echo state networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*, 1263–1272. PMLR.

Grohe, M. 2021. The Logic of Graph Neural Networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, 1–17. IEEE.

Grohe, M. 2023. The Descriptive Complexity of Graph Neural Networks. In *Proceedings of the 38th Annual ACM/IEEE Symposium on Logic in Computer Science*.

Grohe, M.; and Rosenbluth, E. 2024. Are Targeted Messages More Effective? In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, 1–14.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5): 359–366.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Li, Y.; Zemel, R.; Brockschmidt, M.; and Tarlow, D. 2016. Gated Graph Sequence Neural Networks. In *Proceedings of ICLR'16*.

Morgan, H. 1965. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2): 107–113.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 4602–4609.

Otto, M. 1997. Canonization for Two Variables and Puzzles on the Square. *Ann. Pure Appl. Log.*, 85(3): 243–282.

Pfluger, M.; Cucala, D. T.; and Kostylev, E. V. 2024. Recurrent Graph Neural Networks and Their Connections to Bisimulation and Logic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 14608–14616.

Rosenbluth, E.; and Grohe, M. 2025. Repetition Makes Perfect: Recurrent Graph Neural Networks Match Message Passing Limit. arXiv:2505.00291.

Rosenbluth, E.; Toenshoff, J.; and Grohe, M. 2023. Some might say all you need is sum. *arXiv preprint arXiv:2302.11603*.

Rosenbluth, E.; Tönshoff, J.; Ritzert, M.; Kisin, B.; and Grohe, M. 2024. Distinguished In Uniform: Self-Attention Vs. Virtual Nodes. In *The Twelfth International Conference on Learning Representations*.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Selsam, D.; Lamm, M.; Benedikt, B.; Liang, P.; de Moura, L.; Dill, D. L.; et al. 2016. Learning a SAT Solver from Single-Bit Supervision. In *International Conference on Learning Representations*.

Siegelmann, H. T.; and Sontag, E. D. 1992. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, 440–449.

Tönshoff, J.; Kisin, B.; Lindner, J.; and Grohe, M. 2023. One model, any CSP: graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 4280–4288.

Weisfeiler, B.; and Leman, A. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.