How Hard Is It for Message-Passing GNNs to Simulate One Step of the Weisfeiler-Lehman Test?

Guanyu Cui[1] Yuhe Guo[1] Zhewei Wei[1,*] Hsin-Hao Su[2]

[1]Renmin University of China [2]Boston College [*]Corresponding author

NEURAL INFORMATION PROCESSING SYSTEMS

1. Motivation: MPGNNs Match WL at What Cost?

A standard way to characterize MPGNN expressivity is to use the **1-WL test's ability to distinguish graph structures** as an upper bound [1].

MPGNN and WL updates

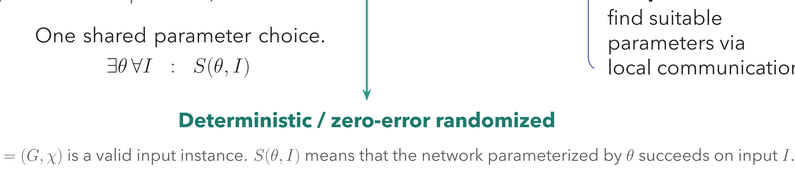
At each layer, an **MPGNN** aggregates messages from a node's neighbors, then updates the node's state by combining its current state with the aggregated messages:

$$\mathbf{h}_v^{(\ell+1)} = \mathrm{UPD}_\theta^{(\ell)}\left(\mathbf{h}_v^{(\ell)},\ \{\{\mathrm{MSG}_\theta^{(\ell)}(\mathbf{h}_u^{(\ell)}) : u \in N(v)\}\}\right).$$

At each iteration, **WL** forms a node's **WL-type** from its current color and the multiset of neighbor colors, then applies an **injective** relabeling:

$$\chi_v^{(t+1)} = \mathrm{HASH}^{(t)}\left(\chi_v^{(t)},\ \{\{\chi_u^{(t)} : u \in N(v)\}\}\right).$$

Running Example. Two consecutive WL steps, starting with color 0 at every node.



Over countable feature spaces, MPGNNs can match WL with **injective** multiset encodings and updates, plus an **injective** readout for graph-level comparison [1].

Observation

An injective map is easy to choose **after** seeing all occurring WL-types.

⇕

An oblivious MPGNN fixes its parameter-selection rule **before** seeing the input.

Research Question

What does it cost for MPGNNs to simulate one WL step?

How do **parameters' dependence on the input instance**, **error guarantees**, and the **color-set size** affect the required resources?

2. Setting

One-step WL simulation

An output coloring $\chi' : V \to [C]$ must satisfy, for every u, v:

$$\chi'_u = \chi'_v \iff (\chi_u, \{\{\chi_x : x \in N(u)\}\}) = (\chi_v, \{\{\chi_y : y \in N(v)\}\}).$$

Here χ is the node coloring at some WL iteration starting from a constant coloring.

Dependence of parameters on the input instance × error guarantee

$I = (G, \chi)$ is a valid input instance. $S(\theta, I)$ means that the network parameterized by θ succeeds on input I.

2. Setting (Cont'd)

Oblivious, deterministic. One parameter choice succeeds on every instance:

$$\exists\theta\ \forall I : \quad S(\theta, I).$$

In the zero-error case, fixing the seed gives a deterministic simulator, so they are equivalent.

Oblivious, bounded-error randomized. One input-independent parameter distribution guarantees success with probability at least $1 - p_f$ on each instance:

$$\exists\mathcal{P}\ \forall I : \quad \Pr_{\theta\sim\mathcal{P}}[S(\theta, I)] \geq 1 - p_f.$$

Requiring $\Pr_{\theta\sim\mathcal{P}}[\forall I : S(\theta, I)] \geq 1 - p_f > 0$ instead gives one parameter choice correct on every input, reducing to deterministic oblivious simulation.

Instance-dependent representation. Suitable parameters exist for each instance; deterministic and bounded-error randomized representation give the same existence guarantee:

$$\forall I\ \exists\theta_I : \quad S(\theta_I, I).$$

Instance-dependent, deterministic adaptation. A shared, input-independent initialization rule and procedure produce a correct d-layer MPGNN evaluation after r graph-local rounds on every instance. The zero-error randomized case is equivalent by the same seed-fixing argument.

Bounded-error randomized adaptation. As in the oblivious case, allowing input-independent randomization and success probability at least $1 - p_f$ on each instance.

3. Main Results: The Cost of One WL Step

Depth d; message width w; precision p bits/coordinate; color-set size $C \geq n$; random bits R; failure probability p_f; nodes n; edges m; diameter D; adaptation rounds r.

Assumptions / model	Bound
Oblivious: deterministic / zero-error randomized	
Lower bound $\log C = o(m)$	$d = \Omega\left(D + \frac{m}{wp}\right)$
Upper bound Rooted, port-aware†	$d = O\left(D + \frac{m \log C}{wp}\right)$
Oblivious: bounded-error randomized	
Large-color upper bound $C \geq \lceil 3n^2/(2p_f) \rceil$	$d = 1,\ wp = \Theta(\log C),\ R = \Theta\left(\log \frac{n \log C}{p_f}\right)$
Any-color upper bound Any $C \geq n$; rooted, port-aware†	$d = O\left(D + \frac{\log C + n \log(n/p_f)}{wp}\right),\ R = \Theta\left(\log \frac{n \log C}{p_f}\right)$
Randomness lower bound $dwp = O(\log C)$, $\log C = o(n)$	$R = \Omega\left(\log \frac{n}{p_f \log C}\right)$
Small-color lower bound $C = \Theta(n)$, $p_f = \Omega(1/\mathrm{poly}(n))$	$d = \Omega\left(D + \min\left\{\log^* n, \frac{n}{wp}\right\}\right)$‡
Instance-dependent: representation	
Upper bound	$d = 1,\ wp = \Theta(\log n)$
Instance-dependent: deterministic / zero-error randomized adaptation	
Lower bound $\log C = o(m)$	$r = \Omega\left(\frac{D}{d} + \frac{m}{dwp}\right)$

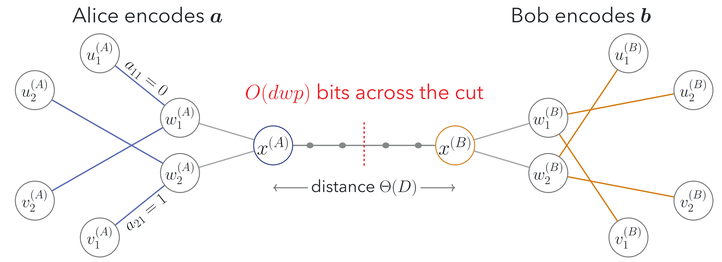
† The marked upper-bound constructions use a rooted, port-aware model allowing a designated root and separate states and messages for each port.

‡ $\log^* n$ is the iterated logarithm. After submission, we obtained the stronger bound $d = \Omega(D + n/(wp))$ for any color-set size $C = \Omega(n)$ with $C = o(n^2/p_f)$.

4. Proof Ideas

Lower bounds: communication across a cut

Inspired by Loukas's communication-complexity perspective [2], we encode binary strings $\boldsymbol{a}, \boldsymbol{b}$ of length $\Theta(m)$ in two graph regions joined by a **single-edge cut**.



Schematic of the core construction. Bit 0 selects $w_i - u_j$; bit 1 selects $w_i - v_j$.

Reduction: Equality → WL. Reusing a one-step simulator for three WL steps makes two distinguished nodes receive the same color exactly when $\boldsymbol{a} = \boldsymbol{b}$, solving $\mathrm{EQ}(\boldsymbol{a}, \boldsymbol{b})$.

We use **communication lower bounds** and **random-bit lower bounds under limited communication** for Equality to obtain corresponding lower bounds for WL simulation.

Upper bounds: CONGEST primitives and hashing

For the **rooted, port-aware upper bounds**, we use standard distributed primitives:

FLOOD build spanning tree → **UPCAST** gather WL-types → **DOWNCAST** broadcast type-to-rank map

FLOOD builds a BFS spanning tree. After UPCAST, the root **deduplicates and sorts** the occurring WL-types to assign ranks. DOWNCAST returns the type-to-rank map, and each node uses its rank as its new color.

For the **bounded-error upper bounds**, we first hash WL-types into short fingerprints. With enough colors, these fingerprints serve directly as labels with failure probability at most p_f, yielding a one-layer construction [3]. With fewer colors, the rooted, port-aware construction gathers and ranks fingerprints, then broadcasts the resulting map.

Takeaways

- **Matching WL has a resource cost.** In many settings, simulating even one WL step can require substantial MPGNN depth or message capacity. The WL expressivity upper bound can thus be costly to attain, and one WL step need not correspond to one MPGNN layer.
- **Randomness and color-set size matter.** With enough colors, bounded error permits one-layer simulation. Randomness can thus enhance MPGNN capabilities under resource constraints.
- **Representation does not imply cheap adaptation.** Although a one-layer instance-specific representation exists, finding suitable parameters through deterministic or zero-error randomized graph-local adaptation can remain costly.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Nos. 92470128 and U2241212) and Renmin University of China.

Main References.

[1] Xu et al. *How Powerful Are Graph Neural Networks?* ICLR 2019.

[2] Loukas. *What Graph Neural Networks Cannot Learn: Depth vs Width.* ICLR 2020.

[3] Aamand et al. *Exponentially Improving the Complexity of Simulating the Weisfeiler–Lehman Test with Graph Neural Networks.* NeurIPS 2022.

Contact: cuiguanyu@ruc.edu.cn zhewei@ruc.edu.cn