Generalized Flow (gflow) as a Loss Function

Luke Hassel

February 23, 2026

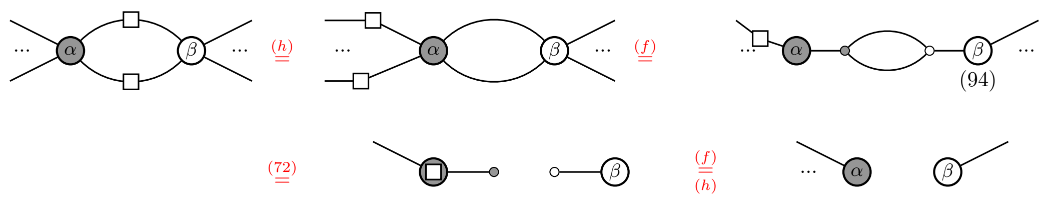
0.1 Graph-like ZX-Diagrams

A ZX-diagram is called *graph-like* [3] if:

- All spiders are Z-spiders (green).
- All edges are Hadamard edges (dashed blue lines).
- There are no self-loops or parallel edges between the same pair of spiders.
- Every input or output is connected to a distinct spider, and no spider is connected to more than one input or output.

The conversion from a ZX-diagram to a graph like ZX-diagrams is done as follows [3]:

1. Convert all X-spiders into Z-spiders by pushing out Hadamards (by using **(h)** of Figure **1**).
2. Cancel all adjacent pairs of Hadamard gates using **(hh)**.
3. Fuse all connected spiders using **(f)**.
4. View the remaining Hadamard gates between spiders as the Hadamard-edges of **(92)**.
5. Remove all self-loops using **(52)** and **(83)**.
6. If two spiders are connected by multiple Hadamard-edges, a variation of the Hopf-rule removes these:



(94)

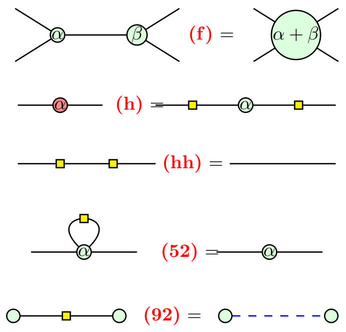


Figure 1: Key ZX-calculus rules used in the transformation: Fusion (f), Color Change (h), Hadamard Cancellation (hh), Loop Removal (52), and Hadamard Edge Definition (92).

After the conversion the ZX-diagram only contains Z-spiders and Hadamard-edges.

0.2 Open Graphs

An *open graph* (G, I, O, λ) is an undirected graph $G = (V, E)$ where $I \subseteq V$ are the inputs and $O \subseteq V$ are the outputs. [1].

A ZX-diagram can be converted to an open graph. Each Z-spider in a graph like ZX-diagram corresponds to a vertex $v \in V$. The input set I consists of all vertices connected to an input wire, and the output set O consists of all vertices connected to an output wire. The edges E of the open graph correspond exactly to the Hadamard edges between the spiders in the diagram [1].

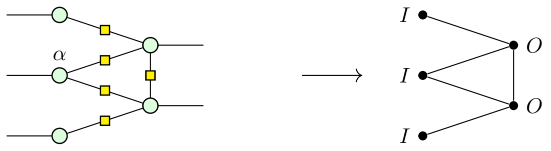


Figure 2: Conversion from a graph-like ZX-diagram (left) to an open graph (right). Spiders with input wires become the input set I, and spiders with output wires become the output set O. Hadamard edges become graph edges.

0.3 Focused Generalized Flow

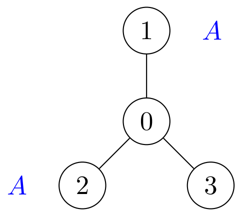
The authors of [1] introduce a special case of the generalised flow definition used in MBQC called focused generalised flow since as a graph property the two notations are equivalent.

(Focused gFlow): Given an open graph $G = (V, E, I, O)$, a *focused gFlow* $(g, \prec)$ on G consists of a function $g : \bar{O} \to 2^{\bar{I}}$ and a partial order $\prec$ on the vertices V of G such that for all $u \in \bar{O}$:

1. $\text{Odd}_G(g(u)) \cap \bar{O} = \{u\}$
2. $\forall v \in g(u), u \prec v$

where $\bar{O} = V \setminus O$ is the set of measured vertices, $\bar{I} = V \setminus I$ is the set of vertices available for the flow map, $2^{\bar{I}}$ is the powerset of $\bar{I}$, and $\text{Odd}_G(A) := \{v \in V(G) \mid |N(v) \cap A| \equiv 1 \pmod 2\}$ is the odd neighbourhood of A.

Example: Calculating Odd$_G(A)$ Consider the star graph G with center vertex 0 and leaves $\{1,2,3\}$. Let the subset $A = \{1,2\}$.



To find $\text{Odd}_G(\{1,2\})$, check the parity of neighbors for each vertex:

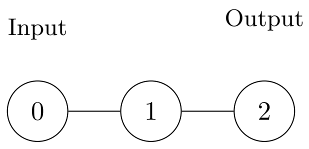
- **Vertex 0:** Neighbors are $\{1,2,3\}$. Intersection with A is $\{1,2\}$. Count $= 2$ (even). $\to 0 \notin \text{Odd}_G(A)$.
- **Vertex 1:** Neighbor is $\{0\}$. Intersection with A is $\emptyset$. Count $= 0$ (even). $\to 1 \notin \text{Odd}_G(A)$.
- **Vertex 2:** Neighbor is $\{0\}$. Intersection with A is $\emptyset$. Count $= 0$ (even). $\to 2 \notin \text{Odd}_G(A)$.
- **Vertex 3:** Neighbor is $\{0\}$. Intersection with A is $\emptyset$. Count $= 0$ (even). $\to 3 \notin \text{Odd}_G(A)$.

In this case, $\text{Odd}_G(\{1,2\}) = \emptyset$. However, if $A = \{0\}$, then all neighbors $\{1,2,3\}$ have exactly one neighbor in A, so $\text{Odd}_G(\{0\}) = \{1,2,3\}$.

1 GFlow Examples

1.1 Example 1: Linear Chain (Has GFlow)

Consider a 3-qubit linear chain with $I = \{0\}$ and $O = \{2\}$:



GFlow:

$$g(0) = \{1\} \qquad\qquad g(1) = \{2\}$$

Partial order: $0 \prec 1 \prec 2$.

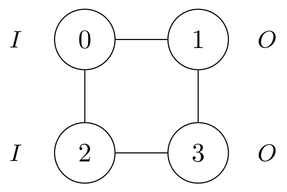
Verification of Condition 1 ($\mathbf{Odd}_G(g(u)) \cap \bar{O} = \{u\}$): Here, the non output set is $\bar{O} = V \setminus O = \{0, 1\}$.

- **For $u = 0$:** $g(0) = \{1\}$. The neighbors of 1 are $\{0, 2\}$, so $\mathrm{Odd}_G(g(0)) = \{0, 2\}$.
  Intersection: $\{0, 2\} \cap \{0, 1\} = \{0\}$. ✓
- **For $u = 1$:** $g(1) = \{2\}$. The neighbor of 2 is $\{1\}$, so $\mathrm{Odd}_G(g(1)) = \{1\}$.
  Intersection: $\{1\} \cap \{0, 1\} = \{1\}$. ✓

Condition 2 ($\forall v \in g(u), u \prec v$) is satisfied as $0 \prec 1$ and $1 \prec 2$.

1.2 Example 2: 2×2 Grid (Has GFlow)

Consider a 4-qubit grid (box) with two inputs $I = \{0, 2\}$ and two outputs $O = \{1, 3\}$:



GFlow:

$$g(0) = \{1\} \qquad\qquad g(2) = \{3\}$$

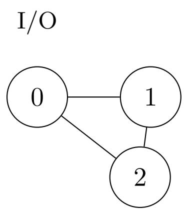
Partial order: $0 \prec 1$ and $2 \prec 3$.

Verification of Condition 1 ($\mathbf{Odd}_G(g(u)) \cap \bar{O} = \{u\}$): The measured set is $\bar{O} = V \setminus O = \{0, 2\}$.

- **For $u = 0$:** $g(0) = \{1\}$. The neighbors of 1 are $\{0, 3\}$, so $\mathrm{Odd}_G(g(0)) = \{0, 3\}$.
  Intersection: $\{0, 3\} \cap \{0, 2\} = \{0\}$. ✓
- **For $u = 2$:** $g(2) = \{3\}$. The neighbors of 3 are $\{1, 2\}$, so $\mathrm{Odd}_G(g(2)) = \{1, 2\}$.
  Intersection: $\{1, 2\} \cap \{0, 2\} = \{2\}$. ✓

1.3 Example 3: Triangle (No GFlow)

Consider a triangle with $I = \{0\}$, $O = \{0\}$ (input and output are the same vertex):



Any assignment creates a cycle: $g(1)$ must use $\{2\}$ and $g(2)$ must use $\{1\}$, requiring both $1 \prec 2$ and $2 \prec 1$, which violates the partial order. **No gflow exists.**

2 Mhalla-Perdrix Algorithm

The standard algorithm for finding a gflow is the algorithm at [2], which operates in $O(n^3)$ time. It relies on solving linear systems over the field $\mathbb{F}_2$.

The core logic involves:

1. Constructing the adjacency matrix of the graph state.
2. Identifying candidate sets using Gaussian elimination over $\mathbb{F}_2$.
3. Iteratively building the gflow layers from outputs back to inputs.

But the algorithm only works with values fixed to 0 or 1 in the g-assigments and because of its iterative nature it is not possible to use it to differentiate.

3 Differentiable Gflow Relaxation

3.1 Loss Function Construction

Since the graph is undirected, only the upper triangular part of the adjacency matrix is needed. Assuming an optimal assignment of g exists, let G^* denote its matrix form.

For example, considering the 2×2 grid from Example 2 with measured qubits $\bar{O} = \{0, 2\}$ and candidate qubits $\bar{I} = \{1, 3\}$, the optimal assignment $g(0) = \{1\}, g(2) = \{3\}$ corresponds to the flow matrix:

$$G^* = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{1}$$

where the rows correspond to $u \in \{0, 2\}$ and columns to $v \in \{1, 3\}$.

3.1.1 Odd Loss

For a vertex u to be in $\bar{O}$, the g-set $g(u)$ must have an odd number of connections to u. This can be expressed as an XOR sum over all vertices v:

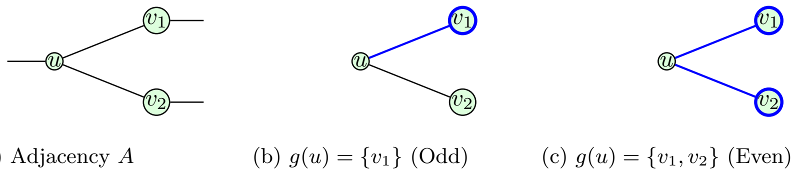
$$\bigoplus_{v \in V} A_{uv} G_{uv} = 1 \tag{2}$$

where $G_{uv} = 1$ if $v \in g(u)$ and 0 otherwise.

Example Parity Check Consider a vertex u with two physical neighbors v_1 and v_2 (so $A_{uv_1} = 1, A_{uv_2} = 1$, and $A_{uv} = 0$ for others). The XOR sum calculates the parity of the selected g-set vertices connected to u:

- If we select $g(u) = \{v_1\}$, then $G_{uv_1} = 1$ and $G_{uv_2} = 0$. The sum is $1 \oplus 0 = 1$. (Condition satisfied, u has an odd number of neighbors in the g-set).
- If we select $g(u) = \{v_1, v_2\}$, then $G_{uv_1} = 1$ and $G_{uv_2} = 1$. The sum is $1 \oplus 1 = 0$. (Condition failed, u has an even number of neighbors in the g-set).

This illustrates how G acts as a filter on the neighborhood defined by A.



(a) Adjacency A (b) $g(u) = \{v_1\}$ (Odd) (c) $g(u) = \{v_1, v_2\}$ (Even)

Figure 3: Open graph fragment illustrating the parity XOR sum. Blue thick edges indicate active neighbors in the g-set ($G_{uv} = 1$).

Note that this captures only the relaxed condition $u \in \mathrm{Odd}_G(g(u))$. The full Focused GFlow definition also requires $\mathrm{Odd}_G(g(u)) \cap \bar{O} = \{u\}$, which is not explicitly enforced.

Example: Focused Condition Violation Consider two non-output vertices $u, u' \in \bar{O}$ that share a common neighbor v, as shown in Figure 4. If the optimization chooses $g(u) = \{v\}$, the condition $u \in \mathrm{Odd}_G(g(u))$ is satisfied since u has exactly one neighbor in its correction set. However, u' also has an odd number of neighbors in the same set $g(u)$, leading to $\mathrm{Odd}_G(g(u)) \cap \bar{O} = \{u, u'\}$. The current loss L_{Odd} does not penalize this because it only evaluates the neighborhood of $g(u)$ at vertex u, failing to enforce that u must be the *unique* non-output vertex in that neighborhood.

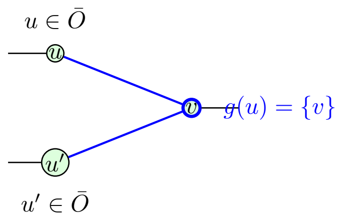


Figure 4: Violation of the focused condition. Selecting $g(u) = \{v\}$ satisfies the parity condition for u, but inadvertently creates an odd neighborhood for u' as well. Since both $u, u' \in \bar{O}$, the condition $\mathrm{Odd}_G(g(u)) \cap \bar{O} = \{u\}$ is violated.

To also allow continuous values (including learning the graph structure $A_{uv} \in [0,1]$), we map the XOR sum to a product over all vertices:

$$\prod_{v \in V} (1 - 2A_{uv}G_{uv}) = -1 \tag{3}$$

In the binary case, this product is -1 when the number of active neighbors (where $A_{uv}G_{uv} = 1$) is odd, and 1 when even. For natural continuous relaxation, the term $1 - 2A_{uv}G_{uv}$ interpolates smoothly: if an edge is absent ($A_{uv} \approx 0$), the factor is close to 1 and does not affect the product. Conversely, if an edge is present ($A_{uv} \approx 1$) but the flow weight is undecided ($G_{uv} \approx 0.5$), the factor vanishes.

Based on this product formulation, the parity loss is defined as the normalized squared distance from the target value -1:

$$\frac{1}{4}\left(\prod_{v \in V} (1 - 2A_{uv}G_{uv}) + 1\right)^2 \tag{4}$$

The term is squared to ensure non-negativity. The factor $1/4$ normalizes the loss: if the condition is satisfied (product is -1), the loss is $\frac{1}{4}(-1+1)^2 = 0$. If violated (product is 1), the loss is $\frac{1}{4}(1+1)^2 = 1$.

Expanding this to all nodes $u \in \bar{O}$ yields:

$$L_{\mathrm{Odd}} = \frac{1}{4} \sum_{u \in \bar{O}} \left(\prod_{v \in V} (1 - 2A_{uv}G_{uv}) + 1\right)^2 \tag{5}$$

3.1.2 Order Loss

To enforce the partial order condition $\forall v \in g(u), u \prec v$ a real-valued "order" value $\tau_v \in \mathbb{R}$ is assigned to each vertex. The discrete condition $u \prec v$ maps to the continuous constraint $\tau_u < \tau_v$.

It is important to note that the values τ are separate variables from the Flow map G and are optimized jointly to satisfy the causal constraints.

This constraint implies that for any active dependency where $G_{uv} \approx 1$ (indicating $v \in g(u)$), the time coordinate of u must be strictly less than that of v. Violations are penalized using a Rectified Linear Unit (ReLU) function with a margin $\epsilon > 0$:

$$L_{\text{order}} = \sum_{u,v} G_{uv} \cdot \text{ReLU}(\tau_u - \tau_v + \epsilon)^2 \tag{6}$$

Here, $\tau_u - \tau_v + \epsilon$ measures the violation. If $\tau_v > \tau_u + \epsilon$ (valid ordering), the term is negative and the ReLU equates to zero. If the ordering is violated or the separation is insufficient, the term is positive, creating a loss proportional to the strength of the assignment G_{uv}.

The margin ϵ is crucial because a standard $\text{ReLU}(\tau_u - \tau_v)$ would only enforce the non-strict inequality $\tau_u \leq \tau_v$. By adding ϵ, we ensure a minimum separation $\tau_v \geq \tau_u + \epsilon$, thereby guaranteeing the strict inequality $\tau_u < \tau_v$ required for a valid order.

The term is squared to penalize larger violations more severely.

Example Calculation Consider two vertices u and v where $G_{uv} = 1$ (meaning $v \in g(u)$), and let the margin be $\epsilon = 0.1$.

- **Valid Ordering:** If $\tau_u = 0.5$ and $\tau_v = 1.0$, then:

  $$\text{ReLU}(0.5 - 1.0 + 0.1) = \text{ReLU}(-0.4) = 0$$

  The loss contribution is zero, as the partial order $u \prec v$ is satisfied with sufficient separation.

- **Invalid Ordering:** If $\tau_u = 0.8$ and $\tau_v = 0.7$ (a backward dependency), then:

  $$\text{ReLU}(0.8 - 0.7 + 0.1) = \text{ReLU}(0.2) = 0.2$$

  The loss contribution is $(0.2)^2 = 0.04$, penalizing the violation.

3.2 Differentiability

For a given graph structure defined by the adjacency matrix A, we seek to identify a valid gflow G and ordering τ by minimizing the total loss:

$$L(A, G, \tau) = L_{\text{Odd}}(A, G) + L_{\text{Order}}(G, \tau) \tag{7}$$

The optimal gflow assignment G^* (and corresponding τ^*) is obtained by optimizing this objective:

$$G^*(A) = \arg\min_{G,\tau} L(A, G, \tau) \tag{8}$$

We then define the function $f(A)$ as the minimum loss value achieved:

$$f(A) = L(A, G^*(A), \tau^*(A)) \tag{9}$$

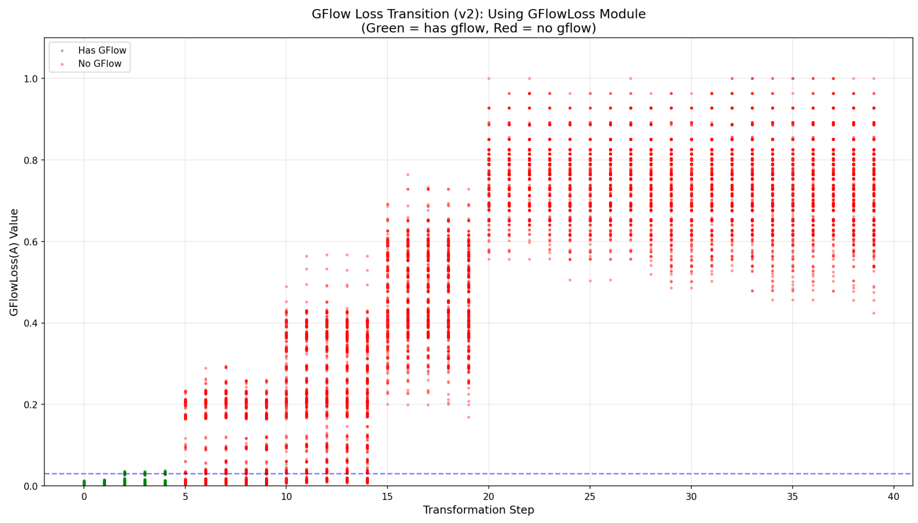


Figure 5: Loss with 6 nodes. Here local complementation is used to create different initial graphs.

Since the loss components are constructed from differentiable operations (element-wise products, summations, and ReLU functions), the resulting function $f(A)$ remains differentiable with respect to the input adjacency matrix A. This allows gradients to be backpropagated through the gflow optimization process to update the underlying graph generator efficiently.

3.3 Simple Gflow Model

Training a simple generative model takes a random vector and outputs an open graph based on the described loss function. This yields gflow in 94% of the open graphs. BFS transformation of the remaining graphs ensures 100% gflow existence.

3.4 Local complementation

Local complementation is used to create different initial graphs for the scatter plots shown in Figure 8. This operation preserves gflow existence [1]. Applying local complementation to a vertex v inverts the adjacency of its neighborhood $N(v)$. For any two neighbors $u, w \in N(v)$, the operation removes the edge (u, w) if it exists, or adds it if it is missing. This transformation leaves the adjacency of v and any vertex outside $N(v)$ unchanged.

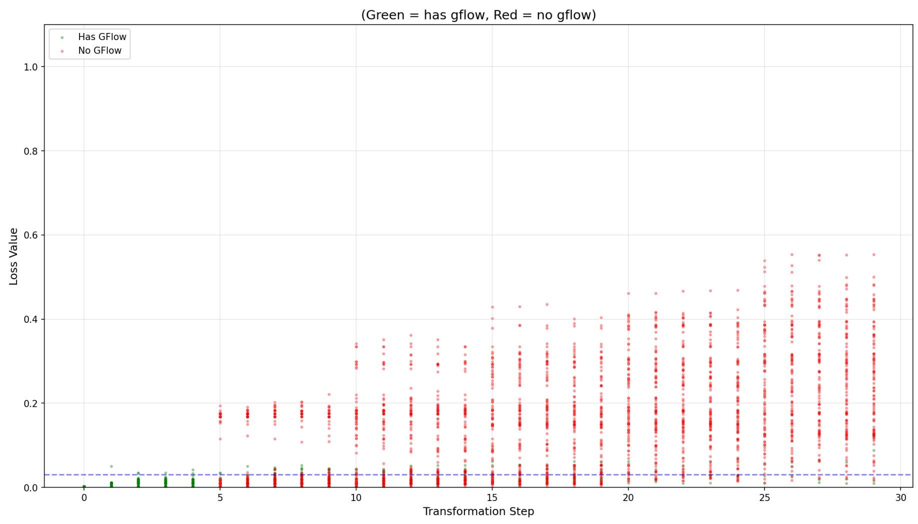


Figure 6: Loss with 20 nodes.

References

[1] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. *Quantum*, 4:279, 6 2020.

[2] Mehdi Mhalla and Simon Perdrix. Finding optimal flows efficiently. *arXiv preprint arXiv:0709.2670*, 2008.

[3] John van de Wetering. Zx-calculus for the working quantum computer scientist. *arXiv preprint arXiv:2012.13966*, 2020.

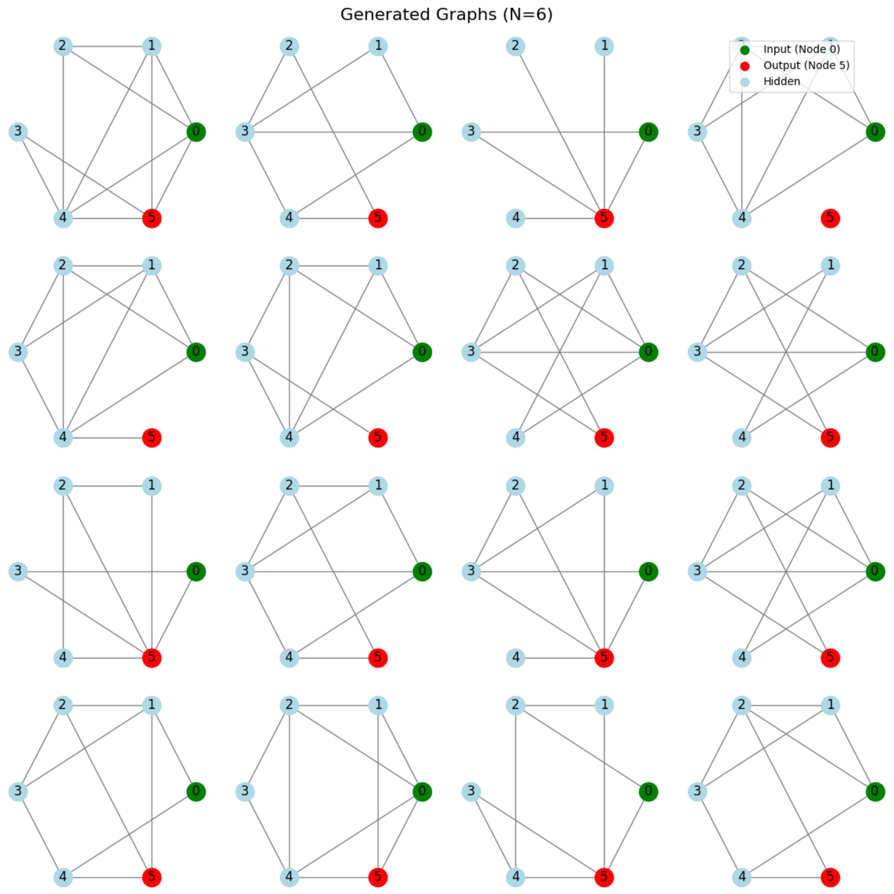


Figure 7: Generator output visualization.

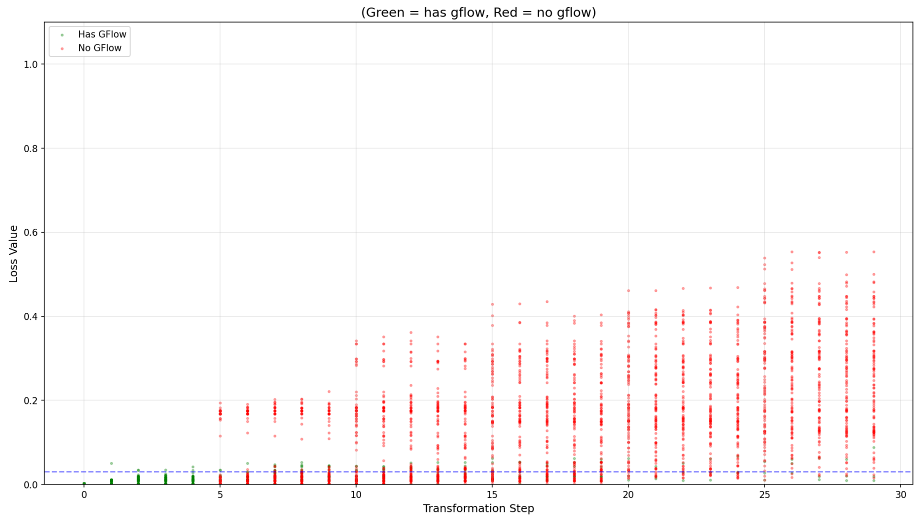


Figure 8: GFlow loss transition scatter plot showing the relationship between step number and loss, with colored gflow.