MODULAR EQUILIBRIUM: STRUCTURE-AWARE SOLVING FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Iteration-based implicit graph solvers—deep equilibrium models and their graph variants—claim to compute an equilibrium, but the equilibrium is *nominal*: the forward pass stops far from its tolerance, yet forcing full convergence changes test performance not at all. We show that the "infinite depth" of these models is carried entirely by cycles, and propose *modular equilibrium*, a fixed-point graph layer whose solver is organized around exactly that structure: decompose the graph into cyclic modules—strongly connected components for directed learned graphs, and bridge (2-edge-connected) decomposition for undirected ones, which we show is the correct analogue since SCCs degenerate on bidirected graphs—solve the fixed point *within* each module with a bounded budget, and propagate *between* modules by bounded feed-forward sweeps along the module graph, with gradients exact for the unrolled computation: bounded where computation is bounded in practice, iterative exactly where feedback lives. We prove that modular propagation is *exact* in the converged limit—it reaches the global fixed point itself (block back-substitution), not an approximation of it—and that its budget follows structural complexity: iterations are spent inside dense cyclic modules, whose truncation tail scales as $1/(1-r)$, and saved on the structurally simple inter-module links. On complex superpixel graphs, a depth-tuned, parameter-matched local baseline only ties global equilibrium solving, while modular solving exceeds both; on small molecular graphs, where a few iterations nearly suffice, it matches global solving at the default coupling and degrades more gracefully as coupling strengthens, while enabling what global solvers cannot express: causal module-importance analysis, which identifies the molecules' macrocyclic rings as the load-bearing units. The advantage tracks the predicted boundary conditions—it vanishes when structural encodings already supply the cyclic structure—and random partitions of the same size distribution destroy the benefit.

1 INTRODUCTION

Graph neural networks whose hidden states are defined by a fixed-point equation—deep equilibrium models (DEQs) and implicit graph neural networks (Bai et al., 2019; Gu et al., 2020; Liu et al., 2021)—have an appealing property: the representation is the solution of an equilibrium system $z = \sigma(u + Az)$, so a single layer is often described as standing in for infinitely many propagation steps. The standard recipe solves this system *globally*: one solver over all nodes—damped fixed-point iteration in the IGNN family (Gu et al., 2020; Liu et al., 2021), or quasi-Newton root finding (Broyden) in the original DEQ (Bai et al., 2019). Quasi-Newton steps are superlinear but carry a low-rank Jacobian over the full $N \times H$ state per step; the graph literature therefore standardized on damped Picard iteration—one matvec per step, trivially batched—run to a tolerance, with gradients from the implicit function theorem (IFT).

This recipe has two structural weaknesses. First, a monolithic solve ignores the graph's own organization. Graphs that arise in chemistry, vision, and beyond are not generic: they consist of cyclic modules (ring systems in molecules, cliques in superpixel adjacencies, feedback motifs in token graphs) connected by acyclic paths. A global contraction treats every edge identically, although cyclic and acyclic edges play very different roles in a fixed-point system—only the cyclic

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ones create the feedback that makes equilibrium solving necessary at all. We make this precise in Section 4.1: equilibrium is a property of cycles. Second, the IFT gradient is exact only *at* the fixed point. Quasi-Newton solvers often do converge; but the widely used iteration-based graph solvers, we will show (Section 6), stop far from it—residuals two orders of magnitude above the solver's tolerance—so the backward pass applies an equilibrium formula at a non-equilibrium point, and truncates its own adjoint iteration on top of that.

We propose *modular equilibrium*, which makes the solver aware of the graph's cyclic structure. The graph is decomposed into modules of mutually reachable nodes. Inside a module, edges form genuine feedback, and we iterate toward equilibrium with a bounded budget; between modules, the module graph is acyclic (or treated as such), and information is propagated by a bounded number of feed-forward sweeps. The whole computation is unrolled, so gradients are exact for the computation actually performed—no implicit assumption, no adjoint truncation. On undirected graphs such as molecules, strongly connected components (SCCs) degenerate to the whole graph; we prove this and replace them with bridge decomposition (2-edge-connected components), the correct undirected analogue of cyclic modules.

Main result. On PascalVOC-SP (official split), modular solving outperforms global solving by $+1.2$ F1; a depth-tuned, parameter-matched GatedGCN (26.0) only *ties* global solving—exceeding that level requires the module structure. Across all three controlled comparisons at identical topology and parameter count—SP, peptides-func, and the strong-coupling regime—modular solving *matches or outperforms global solving in every single one*, with margins from $+0.3$ to $+2.3$ points (Section 5). The margins vary, the direction does not.

Positioning. Our goal is not the leaderboard per se. The strongest LRGB entries stack orthogonal boosters—virtual nodes, positional encodings, larger feature budgets—on top of a fixed solver; our question is upstream of such boosters: *how should the fixed-point solver itself be organized*? We therefore hold topology, parameter count, and protocol fixed in every comparison and vary only the solver. Boosters shrink the solver's margin, as the theory predicts (Appendix C.7).

Our contributions are:

- **A modular equilibrium solver** (Section 3): intra-module bounded equilibrium plus inter-module bounded feed-forward, trained by exact unrolled gradients.
- **Theory** (Section 4): a reframing of DEQ-style fixed points as equilibria of a *cyclic* computation graph—we prove that "infinite depth" is carried entirely by cycles (Proposition 2), a fact the literature has not made explicit; the converged modular limit equals the global fixed point; the bounded variant is a module-aligned Neumann truncation; and we analyze the bias–fidelity trade-off of IFT vs. unrolled gradients.
- **Experiments** (Section 5), where the two LRGB benchmarks play distinct roles. On PascalVOC-SP (complex superpixel graphs) modular solving delivers the *performance* claim: $+1.2$ F1 over global solving, while a depth-tuned GatedGCN (26.0) only ties the global solver, and slower degradation as coupling strengthens. On peptides-func (small molecular graphs) it delivers the *analysis* claim: matched performance, and causal module-importance analysis that identifies macrocyclic rings as load-bearing units—plus the structural ablation (random partitions lose 4.8 points).
- **Analysis** (Section 6): modules are load-bearing—train-time dropping and eval-time causal lesions both cost more than equivalent node-level perturbations; importance concentrates on a few large modules; and—perhaps most useful to the community—a quantified demonstration that the equilibrium in iteration-based implicit graph solvers is nominal.

2 Related Work

Implicit and equilibrium models. DEQs (Bai et al., 2019; 2020) replace depth with a fixed-point equation trained by implicit differentiation; Neural ODEs (Chen et al., 2018) take the continuous limit. On graphs, IGNN (Gu et al., 2020) iterates a convolution to convergence, Lin et al. (2024) learn the solver itself, and monotone variants (Winston & Kolter, 2020; Baker et al., 2024) give

well-posedness guarantees. All solve a *global* fixed point via the IFT; none exploit modular structure. Closest to us, Zhong et al. (2025) learn subgraph representations implicitly for *subgraph-level property prediction*: their fixed point runs over embeddings of task-given subgraphs, ours over node states of topology-discovered cyclic modules; the two task families share no benchmark, so a controlled comparison is not applicable. Cyclic NNs (Yang et al., 2024) use predefined cycles with a forward-forward rule; Liu (2023) learns the network's graph itself—they change what the graph is, we change how the fixed point is computed. A complementary line removes the iteration concern outright: EIGNN (Liu et al., 2021) solves the limit in closed form via eigendecomposition, and (Fu et al., 2024; Gabor et al., 2024) enforce well-posedness by design—our critique (Section 6) targets iteration-based practice. The classical ancestors of our solver are block Gauss–Seidel / block Jacobi iteration and domain decomposition (Saad, 2003), instantiated here with data-given modules. ("Equilibrium propagation" (Scellier & Bengio, 2017) is unrelated.) **Topology: learning, rewiring, long-range passing.** Graph attention (Veličković et al., 2018), adjacency synthesis (Yun et al., 2019), diffusion (Klicpera et al., 2019), rewiring (Topping et al., 2022; Gutteridge et al., 2023), graph transformers (Vaswani et al., 2017; Kreuzer et al., 2021; Rampášek et al., 2022), and virtual nodes (Gilmer et al., 2017; Hu et al., 2020) change the topology; we change the *solver* and hold topology fixed across methods (our QK edges sit in the attention family, identical for all arms; virtual nodes: Appendix C.4). **Decomposition and substructure GNNs.** Cyclic substructures are exploited by junction trees (Jin et al., 2018), hierarchical pooling (Ying et al., 2018), and cycle/clique features (Bouritsas et al., 2023)—as motifs, features, or coarsened computation graphs, never as the *fixed-point solver's* blocks, which is our use. **Long-range message passing and benchmarks.** Long-range communication uses attention (Vaswani et al., 2017; Rampášek et al., 2022; Kreuzer et al., 2021), virtual nodes (Gilmer et al., 2017; Hu et al., 2020), rewiring (Topping et al., 2022; Gutteridge et al., 2023), or infinite-depth schemes (Gu et al., 2020; Liu et al., 2021). We evaluate on LRGB (Dwivedi et al., 2022) against message-passing baselines (Kipf & Welling, 2017; Gilmer et al., 2017; Hamilton et al., 2017; Xu et al., 2019; Battaglia et al., 2018); the coupling-strength study (Section 5.3) follows robustness suites (Gulrajani & Lopez-Paz, 2021; Gui et al., 2022). A reassessment (Tönshoff et al., 2024) shows tuned local message passing closes much of the LRGB gap; our GatedGCN (Bresson and Laurent, 2017) baseline uses our parameter-matched protocol, with the leaderboard score alongside (Table 1). **Structured dropping.** Module dropout relates to stochastic depth (Huang et al., 2016), DropEdge (Rong et al., 2020), DropMessage (Fang et al., 2023), and Standout's adaptive rates (Ba and Frey, 2013); we drop whole *modules* discovered by the graph's cyclic structure. **Graph explanation and attribution.** Post-hoc explainers (GNNExplainer (Ying et al., 2019), SubgraphX (Yuan et al., 2021)) assign heuristic importance scores; our lesion analysis (Section 6) is a *causal ablation* over solver-defined units, which global solvers do not have.

3 METHOD

3.1 SETUP: EQUILIBRIUM GRAPH LAYERS

Let $G = (V, E)$ be a per-sample graph with N nodes and adjacency-like operator $A \in \mathbb{R}^{N\times N}$ with $\max_i \sum_j |A_{ij}| \le r < 1$. A graph equilibrium layer computes

$$z = \sigma(u + Az), \qquad u = W\,\mathrm{LN}(x) + b, \tag{1}$$

where σ is a 1-Lipschitz activation ($\tanh$), x are per-node features (atoms for molecules, color+coordinates for superpixels), LN is LayerNorm, and W, b are shared layer parameters. Since σ is non-expansive and $\|A\|_\infty \le r < 1$, the map $F(z) = \sigma(u + Az)$ is a contraction and (1) has a unique fixed point $z^\star$. The topology A combines given structure (molecular bonds or superpixel adjacency, symmetrically normalized) with learned content-routed edges: $A = \frac{1}{2}A_{\text{given}} + \frac{1}{2}A_{\text{QK}}$, where A_{QK} is a per-row top-k softmax over query–key scores, shared by all compared methods (Section 5). The DEQ baseline solves (1) globally with damped iteration and IFT gradients. Figure 1 gives an overview of our alternative.

3.2 MODULE DECOMPOSITION

Directed graphs. On directed weighted graphs (e.g., learned token graphs), modules are strongly connected components: maximal node sets in which every pair is mutually reachable. The condensation of SCCs is a DAG, giving a natural feed-forward order between modules.

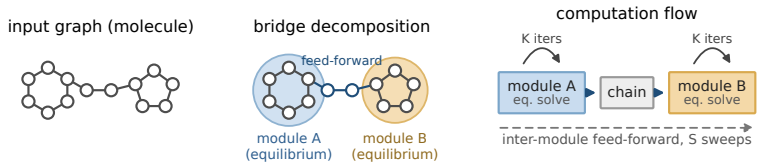


Figure 1: Modular equilibrium: bridge decomposition into cyclic modules, bounded equilibrium inside modules, bounded feed-forward between them.

Undirected graphs. Molecular and superpixel graphs are stored bidirectionally. There, SCCs are useless:

Proposition 1 (Undirected SCC degeneracy)**.** *If the support of A_{given} is symmetric and connected, its only strongly connected component is the whole graph.*

On peptides-func, 199 of the first 200 molecules indeed have exactly one SCC. The correct undirected analogue of a cyclic module is a *2-edge-connected component*: remove all bridges (edges whose removal disconnects the graph) and take the connected components of what remains. Each such component contains at least one cycle (or is a singleton), the components are exactly the cyclic modules (ring systems), and the bridges form the acyclic inter-module skeleton. A peptide averages 127 modules (3.1 non-singleton), a superpixel graph averages more and larger ones. These bidirectional domains are typical applications and the solver's harder case—no topological order, so sweeps are intrinsic (Theorem 2); our experiments focus here.

3.3 Modular equilibrium propagation

Splitting the operator by module. Write $m_i \in \{1, \dots, M\}$ for the module id of node i (from Section 3.2), and let $\mathcal{M}_{ij} = \mathbf{1}[m_i = m_j]$ (the indicator that two nodes share a module). The operator splits as

$$A = \underbrace{A \odot \mathcal{M}}_{A_{\text{in}}:\ \text{intra-module (feedback)}} + \underbrace{A \odot (1 - \mathcal{M})}_{A_{\text{out}}:\ \text{inter-module (feed-forward)}}, \tag{2}$$

with A_{CC} the diagonal block of module C (and $A_{CC'}$ the cross blocks, C' ranging over modules), where $\odot$ is the elementwise product. A_{in} is block-diagonal with one block per module; since submatrices inherit the row bound, each block is itself a contraction.

One sweep. Initialize $z^{(0)} = \sigma(u)$. Sweep $s = 1, \dots, S$ has two steps:

1. *Inter-module feed-forward*: each node collects the current states of the other modules,
$$c^{(s)} = u + A_{\text{out}}\, z^{(s-1)}. \tag{3}$$
2. *Intra-module equilibrium*: within each module, iterate toward a local fixed point with damping $\eta \in (0, 1]$, warm-started from the previous sweep ($z^{(s,0)} = z^{(s-1)}$):
$$z^{(s,k)} = \eta\, \sigma\Big(c^{(s)} + A_{\text{in}}\, z^{(s,k-1)}\Big) + (1 - \eta)\, z^{(s,k-1)}, \quad k = 1, \dots, K. \tag{4}$$

Because A_{in} is block-diagonal, the modules are independent given $c^{(s)}$, so (4) runs for all modules at once as a single batched operation; singleton modules reduce to one evaluation $z_i = \sigma(c_i)$. After S sweeps the layer outputs $z^{(S)}$. The forward budget is fixed in advance: $S(K{+}1)$ matrix–vector products per layer—no convergence check, no input-dependent iteration count. The scheme is a damped block Jacobi iteration (Saad, 2003)—all module blocks updated simultaneously from the previous sweep's state (a sequential block Gauss–Seidel variant processes modules one by one with fresh states)—instantiated with the graph's own cyclic modules and a bounded, unrolled budget.

How far does information travel? One sweep advances one module-hop, so a layer's reach is S hops, compounding to $S{\cdot}L$ across L layers; the learned QK edges additionally shortcut across the module tree. The budget scan (Figure 4a) saturates well before the module-tree depth is reached (mean depth 52 on peptides, 12.5 on PascalVOC-SP; Appendix C.8).

Because the computation is unrolled, backpropagation through it is exact for the computation performed. Unrolling costs memory linear in the budget; sweep-level checkpointing reduces this to $O(S)$ states at $\sim$30% extra compute.

Module dropout. During training we may drop whole modules with probability p: a dropped module keeps its initial state $\sigma(u)$ for that step. This is a capability the modular design adds over global solvers, which have no such unit to drop. Empirically it does *not* work as a regularizer in either domain (Section 5.4)—evidence that the modules are load-bearing.

4 Theoretical Analysis

4.1 Equilibrium is a property of cycles

Read A as information flow: one application of $F(z) = \sigma(u+Az)$ moves information one step along the edges of the computation graph induced by $\mathrm{supp}(A)$. On a DAG the flow is deterministic—it advances along directed paths and terminates after the longest path, so the fixed point of (1) is a finite, feed-forward computation. On a cyclic graph the flow recirculates indefinitely, and one of two things happens: it blows up, or, under our standing contraction condition (σ 1-Lipschitz, $\|A\|_\infty \le r < 1$), it contracts and settles to an equilibrium—the case DEQ-style models study. Such models are usually motivated as "infinitely deep networks"; the flow view makes this precise: the infinite part exists only on cycles.

Proposition 2 (Cycles carry the depth)**.** *Let σ be 1-Lipschitz and $\|A\|_\infty \le r < 1$.*

(i) If $\mathrm{supp}(A)$ is a DAG, the fixed point $z^\star$ of (1) *is computed* exactly *by a single feed-forward pass in topological order—a finite computation whose depth is the longest path of the DAG. No iteration is involved.*

(ii) Iteration, and hence the need for equilibrium solving, arises only on the strongly connected components of $\mathrm{supp}(A)$: the acyclic skeleton between them is always resolved in one pass.

(Proof in Appendix A.3.)

Proposition 2 reframes what an equilibrium layer *is*: not an infinitely deep network, but a cyclic computation graph whose "depth" lives exactly on its cycles. To our knowledge this has not been stated explicitly in the DEQ literature, although it is implicit in the correctness proofs of cyclic-graph propagation of Liu (2023). Modular equilibrium is built on it: feed-forward where the graph is acyclic, equilibrium where it is cyclic. The block back-substitution theorem below is Proposition 2 generalized to mixed cyclic–acyclic graphs.

4.2 Well-posedness

Theorem 1 (Existence, uniqueness, and rate)**.** *Let σ be 1-Lipschitz and $\|A\|_\infty := \max_i \sum_j |A_{ij}| \le r < 1$. Then the map $F(z) = \sigma(u + Az)$ is a contraction with factor r in the norm $\|z\| := \max_i \|z_i\|_\infty$:* (1) *has a unique fixed point $z^\star$, and for $\eta \in (0, 1]$ the damped iteration $z^{(t+1)} = \eta F(z^{(t)}) + (1-\eta)z^{(t)}$ converges linearly with factor $q = 1 - \eta(1-r) < 1$: $\|z^{(t)} - z^\star\|_\infty \le q^t \|z^{(0)} - z^\star\|_\infty$.*

Theorem 1 is standard (Banach); we include the proof in Appendix A for completeness. It underlies both the global DEQ baseline and every intra-module subproblem, since submatrices inherit the row bound: $\|A_{CC}\|_\infty \le r$.

4.3 Exactness in the converged limit

Theorem 2 (Exactness of modular propagation)**.** *Let $z^\star$ be the unique fixed point of* (1)*.* ***(i) Converged limit.*** *For* any *module partition and any A with $\max_i \sum_j |A_{ij}| \le r < 1$, the schedule* (3)–(4) *converges to $z^\star$: it is an asynchronous fixed-point iteration of the global contraction $F(z) = \sigma(u + Az)$ with staleness bounded by one sweep, and such iterations converge to the*

unique fixed point of a sup-norm contraction (Bertsekas–Tsitsiklis; cf. Theorem 1). In particular this holds when A contains learned cross-module QK edges, since $A_{\text{in}} + A_{\text{out}} = A$ regardless of how edges split. ***(ii) Single-pass exactness on DAGs.*** *If the condensation of the module graph is acyclic, a single pass of block back-substitution in topological order recovers $z^\star$ exactly: $z_C = \sigma\big(u_C + A_{CC} z_C + \sum_{C' \prec C} A_{CC'} z_{C'}\big)$. The undirected bridge-block condensation is a bidirectional* tree, not *a DAG; the single-pass schedule does not apply, and exactness there holds in the converged-limit sense of (i).*

(Proof in Appendix A.4.)

Theorem 2 legitimizes the method: modular propagation is a reorganization of the *same* equilibrium, not a different model. The point of the bounded budget is different, however: it turns the exact solver into a structured, predictable computation.

4.4 Bounded budget as module-aligned truncation

Expand the linearization of (1) around the fixed point as a Neumann series $z^\star \propto \sum_{t \geq 0} (DA)^t$ with $D = \sigma'(\cdot)$. A global solver truncated after T iterations keeps the first T terms of this series—a truncation that is blind to the graph's structure. Modular propagation truncates differently: intra-module paths up to length K per sweep are consolidated (modules are iterated to near-convergence), while inter-module propagation is limited to S module-hops per layer. (The exact path support is characterized in Proposition 5, Appendix A.6.)

Why this allocation is the right one. The per-step contraction is uniform—every application of A shrinks the worst-case error by the same r—so the allocation cannot rest on edge types; it rests on *structural complexity*, measured by each sub-operator's spectral radius ρ (walks of length k grow like ρ^k): truncating B after T steps leaves a tail $\sim \rho(B)^T/(1-\rho(B))$. Dense cyclic modules saturate the row-sum budget ($\rho(A_{CC}) \approx r$): thick tail, demanding iteration. Inter-module links are simple: nilpotent on a DAG (zero tail); on undirected graphs $\rho(A_{\text{out}}) \approx w \ll r$ (bridge weights): thin tail. The budget follows structure, not edges (Proposition 6, Appendix A.7).

Proposition 3 (The budget is set by structural complexity)**.** *Consider the linear case and a component B (an intra-module block A_{CC} or the inter-module operator A_{out}) with spectral radius $\rho(B) < 1$. (i) Truncating its Neumann series after T terms leaves a tail of order $\|\text{tail}_B(T)\| \leq \|u\|\, \rho(B)^{T+1}/(1-\rho(B))$; for symmetric non-negative B the bound is attained for suitable u. (ii) Reducing the tail below $\varepsilon \|u\|$ therefore requires $T \geq \log(\varepsilon(1-\rho(B)))/\log \rho(B)$, which grows like $1/(1-\rho(B))$ as $\rho(B) \to 1$. (iii) Instantiations: an inter-module DAG has $\rho(A_{\text{out}}) = 0$ (nilpotent: exact after depth$+1$ sweeps); on undirected graphs $\rho(A_{\text{out}}) \leq w_{\max}$, so inter-module exchange converges at the bridge-weight scale; dense cyclic blocks have $\rho(A_{CC}) \to r$ and demand $\Theta(1/(1-r))$ intra iterations. (Proof in Appendix A.8.)*

Proposition 4 (Bounded-budget error)**.** *Under the conditions of Theorem 1, write the schedule as S sweeps of one inter-module feed-forward step* (3) *followed by K damped intra-module iterations* (4)*, with $q = 1-\eta(1-r)$. Each intra-module iteration contracts the error toward the* proxy *fixed point of the (stale) inter-module input $c^{(s)}$ by at most q, while the proxy itself lies within $\frac{r}{1-r}\, \|z^{(s-1)} - z^\star\|_\infty$ of $z^\star$ (the fixed point of an r-contraction is $1/(1-r)$-Lipschitz in its input). Consequently the global error decays geometrically with the number of* sweeps *S, while block-internal residuals decay with the total intra budget SK; a per-update global contraction at rate q does* not *hold, since inner iterations with stale inter-module input do not contract the global distance independently. The error is not uniform across paths: it concentrates on inter-module paths longer than S hops, while intra-module paths of length $\leq K$ per sweep are consolidated up to q^{SK}. (Proof sketch in Appendix A.5.)*

Empirically, at the default budget (S=6, K=5) the per-layer distance to the global fixed point is $\sim 10^{-2}$–10^{-3}, intra-module residuals 9×10^{-6}–2×10^{-4} (measurement protocol in Appendix E).

4.5 Gradient fidelity

Implicit equilibrium models—whether they iterate (Gu et al., 2020) or use quasi-Newton root finding (Bai et al., 2019)—compute gradients by applying the IFT at the (assumed) fixed point and solving an adjoint system, itself truncated (typically 10 steps). When the forward solve has genuinely converged, this is sound; in the iteration-based graph solvers we study it has not (Section 6), and the

Table 1: **Main results** (test, mean±sample std, 3 seeds). PascalVOC-SP: macro F1, official split (et80: partition computed after dropping the weakest 20% of boundary edges; the solver uses all edges). peptides-func: AP. References: SP—GatedGCN 28.73, GPS 37.48, Exphormer 39.75; peptides-func (official protocol)—GINE 54.98, GatedGCN+RWSE 60.69, GPS 65.35.

Dataset	Method	test
PascalVOC-SP	GatedGCN (matched params, our protocol)	20.99 ± 0.23
	GatedGCN (8-layer, tuned, our protocol)	26.04 ± 0.93
(F1)	eq_adaptive (global)	26.00 ± 0.36
	eq_modular (ours, raw)	26.54 ± 1.93
	eq_modular (ours, et80)	$\mathbf{27.21 \pm 0.93}$
peptides-func	mpnn (finite)	54.35 ± 0.42
(AP)	eq (DEQ, bonds only)	48.25 ± 0.22
	eq_adaptive (global)	56.28 ± 1.66
	eq_modular (ours)	$\mathbf{56.54 \pm 1.37}$

result is an *approximate gradient of an approximate solution*, biased by the forward truncation, the off-equilibrium IFT, and the adjoint truncation. Modular propagation instead uses the *exact gradient of the computation actually performed*. The trade-off is memory: IFT needs $O(1)$ states, unrolling needs $O(SK)$, mitigated by sweep-level checkpointing.

5 Experiments

5.1 Setup

Benchmarks. *PascalVOC-SP* (11,355 superpixel graphs, 21-class node classification, macro F1; official split, edge weights) and *peptides-func* (15,535 molecular graphs, 10-task graph classification, average precision), both from LRGB (Dwivedi et al., 2022). The two datasets span vision and chemistry; the method itself is domain-agnostic (Section 3.2). SP always uses the official split; peptides uses a fixed random 80/10/10 split (seed 0) shared by all methods.

Compared methods. All share the encoder, 4 layers, hidden size 256, optimizer (AdamW, lr 10^{-3}, plateau), 250 epochs, and 3 seeds, and differ only in the propagation layer: **mpnn** (finite), **eq** (global equilibrium on the given graph, the DEQ role), **eq_adaptive** (global equilibrium + learned QK edges), and **eq_modular** (ours: modular propagation on the same topology, S=6, K=5).

5.2 Main results

Table 1 (top) shows the headline comparison: under the official protocol, modular solving with the et80 partition reaches $\mathbf{27.21 \pm 0.93}$, $+1.2$ F1 over the global solver. A parameter-matched GatedGCN reaches only 20.99; even depth-tuned to 8 layers it merely *ties* the global solver (26.04 vs. 26.00). Without the graph the same model collapses to chance (3.9 F1; lift $+23.3$): the structure carries essentially all of the performance.

Module richness determines the benefit. On the dense superpixel mesh, raw bridge decomposition degenerates to one giant module (mean 1.3 per graph): raw modular propagation ties global solving (26.54 vs. 26.00); partitioning after dropping the weakest 20% of boundary edges restores $\sim$350 modules per graph and pulls ahead to the table's best result.

On peptides-func (Table 1, bottom), modular solving again wins at identical topology, with lower variance. The learned QK edges are the dominant accuracy factor (eq $48.25 \rightarrow$ eq_adaptive 56.28), which modular solving preserves. Unrolled gradients buy exactness at 2.0–2.3$\times$ wall time and $O(SK)$ memory, mitigated by checkpointing (Appendix C.3).

Boundary conditions. Both are predicted by the mechanism (Appendix C.7). At the official water-level recipe (5 layers, hidden 300, 500 epochs) the solvers are matched on peptides-func (60.82 vs. 59.76 plain; 64.11 vs. 63.76 with bond gating + RWSE) while modular stays ahead on peptides-struct (0.2914 vs. 0.2995 MAE).

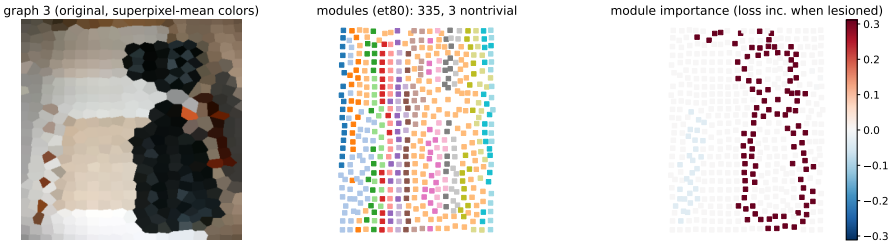
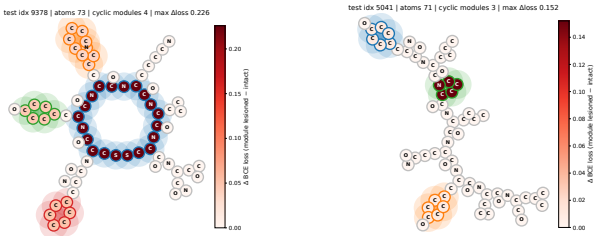


Figure 2: Module attribution on PascalVOC-SP: reconstructed image, et80 module partition, per-module importance (lesion-induced loss increase; red = important). Importance aligns with the foreground object, unsupervised. (More in Appendix E.)



(a) Macrocyclic peptide: the backbone ring (one module) dominates.

(b) Smaller molecule: the central ring dominates.

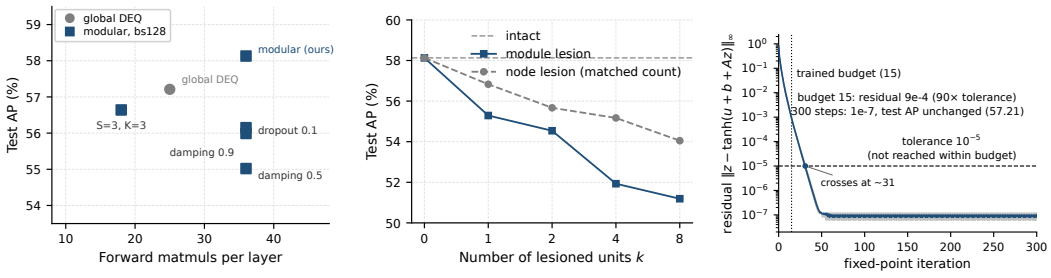
Figure 3: Causal module importance on peptides-func (per-module ΔBCE loss when lesioned; red = important).

5.3 Robustness under strong coupling

Section 4.4 predicts modular propagation should degrade more slowly as coupling strengthens: at r=0.98, 15 global steps remove only ∼19% of the initial error. Across three strong couplings ($r \in \{0.92, 0.95, 0.98\}$, 3 seeds each; Figure 5), modular propagation leads at *every* point (+2.3, +0.3, +1.3 mean AP) and is markedly more stable at the strongest coupling: from r=0.95 to r=0.98 the global solver drops by 1.6 points while modular propagation drops by only 0.6. At weak coupling (r=0.90) the ordering reverses (Appendix C.4).

5.4 Ablations

Four ablations isolate where the benefit comes from (Table 2). First, the solver effect is consistent across the 2 × 2 grid (+0.9 in both topologies). Second—most important—a *random partition of*



(a) (b) (c)

Figure 4: (a) Budget–accuracy frontier. (b) Lesion study: modules hurt 1.5–2.2× more than matched node lesions. (c) Nominal equilibrium: the trained budget stops 90× above tolerance.

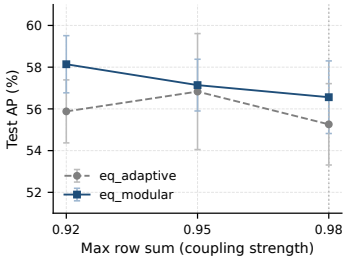


Figure 5: Test AP vs. coupling strength (bs64, 3 seeds).

Table 2: Ablations on peptides-func (test AP, seed 42).

Ablation	Setting	test AP
Topology × solver	global, bonds only	48.20
	modular, bonds only	**49.06**
	global, bonds + QK	57.21
	modular, bonds + QK	**58.13**
Module structure	bridge decomp. (ours)	**58.13**
	random partition	53.38
Module dropout	peptides/SP, 3 settings	-1.8 to -3.0 (hurt)

the same size distribution costs 4.75 points: the cyclic module boundaries carry most of the benefit. Third, the default budget (S=6, K=5) sits in a saturated region (Figure 4a); damping 0.7 is a local optimum. Fourth, module-level dropout always hurts (Table 2): the modules carry real computation (Section 6).

6 Analysis

The equilibrium is nominal (for iteration-based solvers). The trained eq_adaptive solver stops after its 15-iteration budget with residual $\|z - \sigma(u + Az)\| \sim 9 \times 10^{-4}$ at every layer, two orders of magnitude above its tolerance (10^{-5}), which is crossed only at $\sim$31 iterations (Figure 4c). Yet forcing true convergence (300 iterations) leaves test AP unchanged at 57.21: the nominal equilibrium is *irrelevant* to performance. The real computation is a truncated weight-shared iteration whose gradients assume an equilibrium that never happens; modular propagation is honest.

Modules are computational units. Modules are units of the forward computation and can be ablated *causally*: lesioning k modules costs up to 6.9 AP (k=8), 1.5–2.2× more than matched node lesions (Figure 4b). Attribution over 200 test graphs correlates importance with module size (ρ=0.31), not aromaticity (ρ=-0.06); the macrocyclic ring emerges as load-bearing (Figure 3a).

When does modular solving help? All regimes follow one rule: modular solving wins exactly when cyclic structure carries information no other channel supplies. Its value has two channels: *structural information* (cyclic modules as task-relevant units) and *compute allocation* (iterations where walks are densest). The first is substitutable—boosters bypass it and the advantage vanishes; on small graphs the solvers coincide (peptides at water-level recipes). The second matters only under strong coupling (Figure 5). The random-partition control prices the first channel: -4.8 points feature-poor, zero boosted.

7 Conclusion

We recast equilibrium graph computation as a structured, bounded process: equilibrium where the graph has cycles, feed-forward where it does not. At identical topology and parameter count it matches or beats global solving, is more stable at the strongest coupling, and is analyzable at module granularity. It is equally a claim about the literature: what iteration-based solvers call an equilibrium is *nominal*—the forward pass stops far from the claimed fixed point and gradients are taken at an equilibrium that never happens; computing exactly for the computation actually performed is both the honest and the better-posed choice. Modules are also the natural units of analysis: they can be dropped during training and lesioned causally at test time, which is what makes the module-level attributions possible at all. "Infinite depth" lives in cycles, and the equilibrium of iteration-based graph solvers is nominal; its scope is mapped, not assumed (Section 6). Combining modular solving with richer encodings, and mapping the boundary on further datasets and regimes, is future work.

REPRODUCIBILITY STATEMENT

All experiments use fixed seeds (42/43/44), identical protocols across compared methods, and publicly available datasets (Dwivedi et al., 2022). The complete code—data pipelines, modular equilibrium layers, training and analysis scripts, and the exact commands for every table and figure—is included as anonymous supplementary material. Module decomposition is a deterministic, linear-time preprocessing step; bridge decomposition code is included. Hardware: single RTX 4090 per run.

ETHICS STATEMENT

This work studies neural network architectures for graph-structured prediction on public benchmarks (molecular property prediction, image superpixel classification). We see no direct ethical concerns: no human subjects, no sensitive data, no dual-use applications beyond ordinary property prediction.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing, to draft sections of the paper, to retrieve and discuss related work, to assist research ideation and execution (experimental design, protocol choices, and hypothesis refinement), to write parts of the experimental code, and to assist in formalizing parts of the theoretical analysis. All AI-generated code was verified through unit tests, gradient-consistency checks, and controlled ablations; all theoretical statements and reported numbers were checked by the authors. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *NeurIPS*, 2013.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *NeurIPS*, 2019.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *NeurIPS*, 2020.

Joseph Baker, Cheng Wang, Chris Walters, and Rob Brekelmans. Monotone operator theory-inspired message passing. *TMLR*, 2024.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE TPAMI*, 45(1):657–668, 2023.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv:1711.07553*, 2017.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *NeurIPS Datasets and Benchmarks*, 2022.

Taoran Fang, Zhiqing Xiao, Chunping Wang, Jiarong Xu, Xuan Yang, and Yang Yang. DropMessage: Unifying random dropping for graph neural networks. In *AAAI*, 2023.

Guoji Fu, Mohammed Haroon Dupty, Yanfei Dong, and L. W. Sun. Implicit graph neural diffusion networks: Convergence, generalization, and over-smoothing. In *NeurIPS*, 2024.

Mateusz Gabor, Tomasz Piotrowski, and Renato L. G. Cavalcante. Positive concave deep equilibrium models. In *ICML*, 2024.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Fangda Gu, Heng Chang, Wenwu Zhu, Siamak Rasekh, and Minsu Cho. Implicit graph neural networks. In *ICML*, 2020.

Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. GOOD: A graph out-of-distribution benchmark. In *NeurIPS*, 2022.

Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *ICLR*, 2021.

Benjamin Gutteridge, Xiaowen Dong, Michael Bronstein, and Francesco Di Giovanni. DRew: Dynamically rewired message passing with delay. In *ICML*, 2023.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

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

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *NeurIPS*, 2021.

Junchao Lin, Zenan Ling, Zhanbo Feng, Jingwen Xu, and Minxuan Liao. IGNN-Solver: A graph neural solver for implicit graph neural networks. *arXiv:2410.08524*, 2024.

Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. EIGNN: Efficient infinite-depth graph neural networks. In *NeurIPS*, 2021.

Xinshun Liu. Automated architecture synthesis for arbitrarily structured neural networks. *arXiv:2306.02157*, 2023.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 2017.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *ICML*, 2023.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? Reassessing the long-range graph benchmark. *TMLR*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Ezra Winston and J. Zico Kolter. Monotone operator equilibrium networks. In *NeurIPS*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Liang Wei Yang, Hengrui Zhang, Weizhi Zhang, Zihe Song, Jing Ma, Jiawei Zhang, and Philip S. Yu. Beyond directed acyclic computation graph with cyclic neural network. *arXiv:2402.13041*, 2024.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating explanations for graph neural networks. In *NeurIPS*, 2019.

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *ICML*, 2021.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In *NeurIPS*, 2019.

Yongjian Zhong, Liao Zhu, Hieu Vu, and Bijaya Adhikari. Implicit subgraph neural network. In *ICML*, 2025.

A Full Proofs

A.1 Proof of Theorem 1

Proof. Write $F(z) = \sigma(u + Az)$. For any z, z' and any node i,

$$|F(z)_i - F(z')_i| = \left|\sigma\Big(u_i + \textstyle\sum_j A_{ij} z_j\Big) - \sigma\Big(u_i + \textstyle\sum_j A_{ij} z'_j\Big)\right| \tag{5}$$

$$\leq \left|\textstyle\sum_j A_{ij}(z_j - z'_j)\right| \leq \displaystyle\sum_j |A_{ij}|\, \|z - z'\|_\infty \leq r\, \|z - z'\|_\infty, \tag{6}$$

using 1-Lipschitzness of σ in the first inequality and the row bound in the last. Taking the max over i gives $\|F(z) - F(z')\|_\infty \leq r\|z - z'\|_\infty$ with $r < 1$, so F is a contraction and the Banach fixed-point theorem yields a unique $z^\star$.

For the damped iteration $z^{(t+1)} = \eta F(z^{(t)}) + (1-\eta)z^{(t)}$, $\eta \in (0, 1]$: since $z^\star$ is also a fixed point of the damped map,

$$\|z^{(t+1)} - z^\star\|_\infty \leq \eta\, \|F(z^{(t)}) - F(z^\star)\|_\infty + (1-\eta)\, \|z^{(t)} - z^\star\|_\infty \tag{7}$$

$$\leq \big(\eta r + (1-\eta)\big)\, \|z^{(t)} - z^\star\|_\infty = q\, \|z^{(t)} - z^\star\|_\infty, \tag{8}$$

with $q = 1 - \eta(1 - r) < 1$. Iterating gives the claimed rate. □

A.2 Proof of Proposition 1 (undirected SCC degeneracy)

Proof. If the support of A_{given} is symmetric, then every edge is bidirectional, so mutual reachability coincides with connectivity: for any two nodes u, v in the same connected component there is an undirected path between them, which is a directed path in each direction. Hence every connected component is strongly connected, and the SCCs are exactly the connected components. If the graph is moreover connected, the only SCC is the whole graph. □

On peptides-func, 199 of the first 200 molecules have exactly one SCC; the exception is a disconnected graph (two components).

A.3 Proof of Proposition 2 (cycles carry the depth)

Proof. (i) Read each application of A as information flowing one step along the edges: node i receives only from its predecessors, since its equation $z_i = \sigma(u_i + \sum_{j \prec i} A_{ij} z_j)$ involves no other nodes. On a DAG, process the nodes in topological order: when i is processed, every node that can send information to it has already settled to its final value, so the flow front advances deterministically and terminates after the longest path—one pass, no iteration. The resulting values satisfy (1) exactly; by Theorem 1 the fixed point is unique, so this pass computes $z^\star$. (ii) Contracting each SCC to a point yields a DAG (the condensation): inter-module information likewise flows along a DAG, so (i) applies at the level of modules, and the acyclic skeleton between SCCs is always resolved in one pass. □

In the language of Section 4.1: the "infinite depth" of an equilibrium model is, structurally, nothing but the graph's cyclic part—rings *together with everything woven through them* (chords, fused and spiro rings): precisely the cyclic modules of Section 3.2. The acyclic remainder is a finite, one-pass propagation.

A.4 Proof of Theorem 2 (exactness of modular propagation)

Proof. (ii) Order the modules $C_1, \ldots, C_M$ so that all edges between distinct modules go from earlier to later modules (a topological order of the condensation DAG). We prove by induction over i that z_{C_i} produced by the block solve equals the restriction of $z^\star$ to C_i. For $i = 1$ (a source module), there are no incoming inter-module edges, so the block equation is $z_{C_1} = \sigma(u_{C_1} + A_{C_1 C_1} z_{C_1})$, which is exactly the restriction of (1) to C_1; by Theorem 1 applied to the submatrix $A_{C_1 C_1}$ (contraction, $\|A_{C_1 C_1}\|_\infty \leq r$), the solution is unique and equals $z^\star|_{C_1}$. For the induction step, assume $z_{C_j} = z^\star|_{C_j}$ for all $j < i$. The block equation for C_i reads

$$z_{C_i} = \sigma\Big(u_{C_i} + A_{C_i C_i} z_{C_i} + \textstyle\sum_{j<i} A_{C_i C_j}\, z^\star|_{C_j}\Big), \tag{9}$$

whose right-hand side coincides with the restriction of (1) to C_i, because all inter-module contributions to rows in C_i come from modules earlier in the order. Again by contraction, the block solution is unique, hence $z_{C_i} = z^\star|_{C_i}$. This covers all modules, giving (ii). (i) Each sweep's first intra-module update reads $\eta\,\sigma(u + A_{\text{in}} z + A_{\text{out}} z) + (1-\eta) z$ when started from the sweep's own previous iterate—exactly a damped step of the global contraction F; the remaining intra updates use a stale inter-module input (bounded by one sweep), so the schedule is an asynchronous iteration of F with bounded staleness, which converges to $z^\star$ by the Bertsekas–Tsitsiklis asynchronous convergence theorem for sup-norm contractions. Numerically, modular propagation with a large budget matches the global solve to 3×10^{-7} (max abs. difference). □

A.5 Derivation of the bounded-budget error (Proposition 4)

Proof sketch. Two separate statements are at play; we do not claim a per-update global contraction. (i) *Convergence.* Each sweep's first intra-module update, started from the sweep's own previous iterate $z^{(s-1)}$, reads $\eta\,\sigma(u + A_{\text{in}} z^{(s-1)} + A_{\text{out}} z^{(s-1)}) + (1-\eta) z^{(s-1)}$—exactly the damped global step T_η, which contracts the sup-norm error by q (Appendix A). The remaining intra-module updates use the stale input $c^{(s)}$, so the schedule is an asynchronous fixed-point iteration of the global contraction

F with staleness bounded by one sweep; such iterations converge to the unique fixed point of a sup-norm contraction (Bertsekas–Tsitsiklis), and the envelope argument behind that theorem contracts the sup-norm error by q over each cycle in which every block is updated at least once—here, once per sweep—giving geometric decay in S. No uniform per-update rate holds at large r: inner updates with stale inter-module input do not contract the global distance independently. (ii) *Error structure.* Within one sweep, intra-module blocks are iterated K times against the proxy fixed point of $c^{(s)}$, and the proxy map $z \mapsto \sigma(c^{(s)} + A_{\text{in}} z)$ is itself a contraction with factor q, so block-internal residuals decay like q^{SK}. The proxy is $1/(1-r)$-Lipschitz in its input, hence within $\frac{r}{1-r}\|z^{(s-1)} - z^\star\|_\infty$ of $z^\star$. Inter-module information advances only once per sweep (through $c^{(s)}$), so after S sweeps it has traveled at most S module-hops; paths longer than S hops in the condensation are the least consolidated. This is the sense in which the error "concentrates on long inter-module paths"—and why weak inter-module coupling (bridges are single edges, $\|A_{\text{out}}\|_\infty \ll r$) governs the practical convergence speed. □

A.6 Pathwise view: the Neumann truncation is module-aligned

The phrase "module-aligned truncation of the Neumann series" (Section 4.4) can be made exact. Consider the linear case $\sigma = \mathrm{id}$ (equivalently, the linearization at the fixed point, $A \leftarrow DA$). The global fixed point is the Neumann series

$$z^\star = (I - A)^{-1} u = \sum_{t \geq 0} A^t u, \tag{10}$$

and $A^t u$ sums the weights of all length-t directed paths in $\mathrm{supp}(A)$. A global solver stopped after T iterations keeps exactly the paths of length $\leq T$: a uniform cutoff, blind to the graph's structure.

Proposition 5 (Path support of modular propagation)**.** *In the linear undamped case, modular propagation with budget (S, K) computes a partial Neumann sum whose support is* contained in *the set of paths that use at most S inter-module edges and whose intra-module runs have length at most SK (up to the initial-state term $z^{(0)}$). The containment can be strict, because sweep interleaving constrains the order in which intra- and inter-module edges may appear along a path. Damping rescales coefficients but not the support; the warm-started implementation extends each intra-module run by at most K edges per sweep.*

Proof sketch. Write $A = A_{\text{in}} + A_{\text{out}}$ (intra-/inter-module parts). One sweep applies $z \leftarrow \sum_{k=0}^{K-1} A_{\text{in}}^k (u + A_{\text{out}} z) + A_{\text{in}}^K z$: the first term injects source paths and extends existing paths by one inter-module hop followed by an intra-module run of length $\leq K-1$; the second extends existing paths by K intra edges without new hops. Induction over sweeps: after sweep s the iterate contains only paths with $\leq s$ inter-module edges, each contributing its monomial $A^{\text{path}} u$ with coefficient 1. Inter-module hops can only enter through A_{out}, once per sweep, giving the budget S; intra-module runs grow by at most K per sweep, giving SK. Every computed path satisfies the stated bounds, but not every path satisfying the bounds is generated: e.g., with $(S, K) = (1, 2)$ the path $A_{\text{in}} A_{\text{out}} A_{\text{in}}$ satisfies both yet cannot occur, since a single sweep starting from $z^{(0)} = u$ (which contains no edges) applies intra-module edges only after the inter-module hop, never on both sides of it. □

So the two truncation schemes cut the same series along different geometries: global iteration cuts by *path length* ($\leq T$), modular propagation cuts *within the structural envelope* ($\leq S$ hops between modules, $\leq SK$ steps inside them). Since long-range information flows along inter-module hops while feedback lives in intra-module runs, the module-aligned cut spends the budget where the truncation tail is thick (cycles, tail $\sim r^K/(1-r)$) and saves where it is thin (DAG hops, $\sim r^S$ with no cyclic amplification). For nonlinear σ the statement holds for the linearization at the fixed point, i.e. for the first-order sensitivity of $z^\star$ to u; higher-order terms mix paths.

A.7 Spectral budget allocation

The per-step contraction factor of the iteration $z \leftarrow \sigma(u + Az)$ is uniform (r; cf. the proof of Theorem 1), so the budget allocation must rest on structure, not edge types. The relevant mea-

sure is each sub-operator's spectral radius: truncating B after T terms leaves a tail bounded by $\|u\|\,\rho(B)^{T+1}/(1-\rho(B))$, and ρ is the growth rate of walks (walks of length k number $\sim \rho^k$).

Proposition 6 (Budget allocation is spectral, not per-edge)**.** *Let A be symmetric (undirected graphs), split as $A = A_{\text{in}} + A_{\text{out}}$ with intra-/inter-module parts. Then:*

(i) $\rho(A_{\text{out}}) = \|A_{\text{out}}\|_2 \le w_{\max} := \max_i \sum_{j:\, m_j \neq m_i} |A_{ij}|$, the largest per-node total inter-module weight. Truncating the isolated inter-module series $\sum_k A_{\text{out}}^k u$ after S terms leaves a tail of at most $\|u\|_2\, \rho(A_{\text{out}})^{S+1}/(1-\rho(A_{\text{out}}))$.

(ii) Each intra-module block satisfies $\rho(A_{CC}) = \|A_{CC}\|_2 \le r$, and dense cyclic blocks saturate this bound ($\rho(A_{CC}) \approx r$).

(iii) Hence the ratio of tail thicknesses, $\frac{1-\rho(A_{\text{out}})}{1-\rho(A_{CC})}$, can reach one to two orders of magnitude for strongly coupled modules connected by weak bridges (e.g., r=0.98, w=0.2: 50 vs. 1.25). The budget should follow ρ, not edge counts.

Proof. (i) A_{out} is symmetric, so $\|A_{\text{out}}\|_2 = \rho(A_{\text{out}})$; by Gershgorin, $\rho(A_{\text{out}}) \le \max_i \sum_j |(A_{\text{out}})_{ij}| = w_{\max}$. The tail bound is the geometric series of $\|A_{\text{out}}^k u\|_2 \le \rho^k \|u\|_2$. (ii) Same argument applied to the diagonal blocks, whose row sums reach r. (iii) Direct substitution. □

Why the intra-module share must be larger. Write the Neumann series of the linearized system as a sum over walks. For any sub-operator B, walks of length k in its block number $\sim \rho(B)^k$, so truncating after T terms leaves a tail of order

$$\text{tail}_B(T) \asymp \frac{\rho(B)^{T+1}}{1-\rho(B)}, \tag{11}$$

the geometric accumulation of all discarded walks. Two structural facts make the two components sit at opposite ends of this scale. First, intra-module blocks are *dense cyclic structures* (rings *with chords*): every node's row-sum budget r is essentially spent on intra-module edges, so $\rho(A_{CC}) \approx r$, and walks of every length exist in large numbers—the tail is thick and only many iterations ($S{\times}K$) shrink it. Second, inter-module links are *structurally simple*: on a DAG, A_{out} is nilpotent ($\rho = 0$, tail exactly zero after depth+1 sweeps); on undirected graphs, A_{out} is symmetric with row sums equal to single bridge weights $w \ll r$, so $\rho(A_{\text{out}}) \approx w$ and (11) gives a thin tail $w^{S+1}/(1-w)$.

Accuracy vs. coverage. The thin tail means few sweeps suffice for the *accuracy* of what propagates; it does not speak to *coverage*: one sweep advances one module-hop, so information crosses the bridge-block tree only after d hops ($d = 52$ on peptides; Appendix C.8), and with $S{<}d$ distant modules never exchange within a layer at all. The two constraints are separate: the tail governs how much of each walk's weight survives, coverage governs which modules hear anything. In practice coverage is restored by layer compounding ($S{\cdot}L$ hops across L layers) and by the learned QK shortcuts, and the budget scan shows the task saturates before full coverage is needed (Appendix C.8).

The allocation rule. A rational budget sets T per component so that $\text{tail}_B(T)$ falls below the task's noise level: solve (11) for T, $T_B \approx \log(\text{noise}\cdot(1-\rho(B)))/\log\rho(B)$. Since $\rho(A_{CC}) \gg \rho(A_{\text{out}})$, T_{in} must be large (iterate modules to near-equilibrium) while T_{out} stays small (a few sweeps). The allocation is therefore *structural*: it follows the spectral radius—the density of cyclic walks—of each component, not the count or type of its edges. Note the contrast with the single-step view: one application of A contracts the worst-case error by the same r everywhere, so *no* per-edge analysis can justify the split; only the accumulated walk structure (the tail (11)) can.

On directed learned graphs the inter-module condensation is a DAG, so A_{out} is nilpotent and the tail vanishes exactly (Appendix A.6); Proposition 6 is the undirected counterpart.

A.8 Proof of Proposition 3

Proof. (i) For symmetric B, expand u in an orthonormal eigenbasis of B: $u = \sum_i \alpha_i v_i$, so $\|B^k u\|_2^2 = \sum_i \alpha_i^2 \lambda_i^{2k}$ and the tail $\sum_{k>T} B^k u$ has norm bounded by $\|u\|_2 \sum_{k>T} \rho(B)^k =$

$\|u\|_2\,\rho(B)^{T+1}/(1-\rho(B))$. For non-negative symmetric B, $+\rho(B)$ is itself an eigenvalue (Perron–Frobenius), and taking $u = v_{\max}$ (a top eigenvector) attains the bound. For general B, Gelfand's formula gives $\|B^k\| \le C_0\,\rho(B)^k$, the same bound up to the constant. (ii) Rearranging $\rho(B)^T/(1-\rho(B)) \le \varepsilon$ gives $T \ge \log(\varepsilon(1-\rho(B)))/\log\rho(B)$; with $\log\rho(B) \approx -(1-\rho(B))$ as $\rho(B) \to 1$ this is $\Theta(1/(1-\rho(B)))$ for fixed ε. (iii) Nilpotency is immediate from the DAG structure (walks are simple paths of length at most the depth). The bridge-weight bound is Proposition 6(i). Dense cyclic blocks saturate the row-sum budget, so $\rho(A_{CC}) \to r$ and (ii) applies with $\rho(B) = r$. □

A.9 A TWO-NODE TOY EXAMPLE: CHAINS SUBSTITUTE, CYCLES ITERATE

The difference between acyclic and cyclic parts is visible already on two nodes. For a chain, $z_1 = \sigma(u_1)$, $z_2 = \sigma(0.5z_1 + u_2)$: values are obtained by substitution, exactly, in one pass—a coverage problem. For a cycle, $z_1 = \sigma(0.5z_2 + u_1)$, $z_2 = \sigma(0.5z_1 + u_2)$: neither node can be evaluated first, and one iterates $z_i \leftarrow \sigma(0.5z_j + u_i)$; each round multiplies the error by ≈ 0.5, so after K rounds the residual is $\approx 0.5^K$—a convergence problem. Chains are exact once visited; cycles need iteration, and their truncation error decays only geometrically with the iteration count (with the $1/(1-r)$ amplification discussed in Section 4.4). This is why the budget in (3)–(4) is spent asymmetrically.

A.10 GRADIENT BIAS OF TRUNCATED IMPLICIT DIFFERENTIATION

For the global solver, define the forward iteration $z^{(t+1)} = \eta\sigma(u + Az^{(t)}) + (1-\eta)z^{(t)}$, stopped after T_f steps at $\hat{z}$, and the IFT gradient obtained by iterating the adjoint map $v^{(t+1)} = D\big(g + A^\top v^{(t)}\big)$ (damped) for T_b steps, where $D = \sigma'(u + A\hat{z})$. The bias of the resulting gradient estimator $\widehat{\nabla}$ relative to the exact gradient of the true fixed point $z^\star$ decomposes as:

1. **Forward truncation**: $\hat{z} \neq z^\star$, so the Jacobian D and the implicit system are evaluated at the wrong point; $\|\hat{z} - z^\star\| = O(q^{T_f})$ (Theorem 1).
2. **Model-form error**: the IFT formula is derived under the assumption $\hat{z} = z^\star$; applying it at $\hat{z} \neq z^\star$ is a first-order error in the implicit system itself.
3. **Adjoint truncation**: the adjoint solve is stopped after T_b steps, contributing another geometrically decaying error (asymptotic rate at most q, since $\rho(\eta DA^\top + (1-\eta)I) \le \eta\rho(DA^\top) + (1-\eta) \le q$; exactly $O(q^{T_b})$ for symmetric A).

In our measurements on peptides-func, $T_f{=}15$ leaves residuals $\sim 9{\times}10^{-4}$ (two orders of magnitude above the nominal tolerance), so terms (1) and (2) are the dominant ones; see Section 6. The unrolled alternative used by modular propagation has zero terms of all three kinds: it differentiates exactly the computation that was performed, at the cost of $O(SK)$ memory (reducible to $O(S)$ by sweep-level checkpointing).

A.11 BACKWARD PASS OF MODULAR PROPAGATION (EXACT)

Since every step of the forward computation is a matrix–vector product plus a pointwise nonlinearity, the backward pass is the mirror image of the forward pass—same budget, no iterative adjoint solve, no tolerance, no equilibrium assumption. We give the recurrences explicitly. Consider one sweep (3)–(4) of one layer, and write $g^{(k)} = \partial L/\partial z^{(k)}$ for the upstream gradient arriving at the k-th intra-module iterate. For the damped update $z^{(k)} = \eta\,\sigma(w) + (1-\eta)\,z^{(k-1)}$ with $w = c^{(s)} + A_{\text{in}}z^{(k-1)}$, the chain rule gives, stepping $k = K, K-1, \ldots, 1$:

$$\frac{\partial L}{\partial w} = \eta\, g^{(k)} \odot \sigma'(w), \tag{12}$$

$$\frac{\partial L}{\partial z^{(k-1)}} \mathrel{+}= (1-\eta)\, g^{(k)} + A_{\text{in}}^\top \frac{\partial L}{\partial w}, \tag{13}$$

$$\frac{\partial L}{\partial A_{\text{in}}} \mathrel{+}= \frac{\partial L}{\partial w}\,(z^{(k-1)})^\top, \qquad \frac{\partial L}{\partial c^{(s)}} \mathrel{+}= \frac{\partial L}{\partial w}, \tag{14}$$

where $\odot$ is elementwise multiplication and $+=$ denotes accumulation (each $z^{(k-1)}$ also receives gradient from the next intra step). After the intra loop, the inter-module step $c^{(s)} = u + A_{\text{out}}z^{(s-1)}$

contributes

$$\frac{\partial L}{\partial A_{\text{out}}} \mathrel{+}= \frac{\partial L}{\partial c^{(s)}} \left(z^{(s-1)}\right)^{\top}, \qquad \frac{\partial L}{\partial u} \mathrel{+}= \frac{\partial L}{\partial c^{(s)}}, \qquad \frac{\partial L}{\partial z^{(s-1)}} \mathrel{+}= A_{\text{out}}^{\top} \frac{\partial L}{\partial c^{(s)}}, \tag{15}$$

and the sweep repeats in reverse for $s = S, \ldots, 1$. Gradients for the QK-completion parameters flow through the softmax/top-k construction of A_{QK} in the standard way. Every operation is a matmul by a (transposed) matrix or a pointwise map, so the backward pass has the same $S(K+1)$ budget as the forward pass (a factor of 2 in matmul count, one per operand) and is exact by construction. This is the concrete sense in which our gradient avoids all three bias terms of truncated IFT (Section A.10): there is no adjoint fixed point to truncate, and no equilibrium assumption to violate.

A.12 Remark: batched sweeps vs. sequential topological solve

The $A_{\text{in}}/A_{\text{out}}$ design serves two purposes: a structural principle (feedback needs iteration, feed-forward does not) and a GPU-friendly batched realization (A_{in} is block-diagonal, so all modules of all graphs in a batch are processed by a single matrix multiplication per step).

A classical alternative is the *sequential* topological solve of Liu (2023): process modules one by one in condensation order, solving each intra-module system to convergence given the already-converged upstream modules. The two coincide exactly when intra-module iterations converge and the number of sweeps S reaches the depth of the module DAG—the inter-module operator A_{out} is nilpotent of index at most the DAG depth plus one, so after that many Jacobi sweeps the batched computation equals the sequential one, and by Theorem 2 both equal the global fixed point.

At finite budgets they differ: the sequential version is exact in one pass but does not parallelize across modules (a Python-level loop over modules per sample); the batched version trades exactness of long-range inter-module propagation for full parallelism and a fixed, input-independent budget. Our choice is the latter, and the accuracy cost of that trade is what Figure 4a quantifies.

A.13 Remark: solver choice—fixed-point iteration vs. quasi-Newton

The original DEQ (Bai et al., 2019) solves both the forward and the adjoint system with Broyden's quasi-Newton method, which converges superlinearly near the solution. The graph line (IGNN, EIGNN (Gu et al., 2020; Liu et al., 2021)) instead uses plain damped fixed-point iteration. The reasons are structural, not accidental: (i) these models are *designed* around a contractive map ($\|A\|_\infty < 1$ via the row-sum cap), under which iteration is guaranteed to converge linearly and predictably, while quasi-Newton methods carry no convergence guarantee without line-search safeguards and can diverge on ill-conditioned maps; (ii) graph learning is per-sample: every sample is a different fixed-point problem, and quasi-Newton needs a separate secant state per sample, which batches poorly across heterogeneous graphs, whereas damped iteration is a single batched matrix multiplication for the whole batch; (iii) iteration keeps the per-step cost and implementation minimal.

Quasi-Newton's advantage—fast terminal convergence—does not change the structural picture of this paper: the iteration count remains input-dependent and the budget unbounded; and the locus of equilibrium (cycles) is unaffected by the root finder. Our nominal-equilibrium measurement (Section 6) is scoped to the iteration-based family that the graph literature actually uses. Note finally that our modular propagation is orthogonal to the local solver: the intra-module iteration (4) could equally be accelerated by Anderson acceleration or Broyden steps per block; we use damped iteration for batching simplicity and a fixed budget.

B Experimental Details

B.1 Datasets

Module decomposition is computed once per dataset (6 s for all 15,535 peptides) by a linear-time Tarjan bridge finder; module ids are stored as a per-node attribute.

Table 3: Dataset statistics.

	peptides-func	PascalVOC-SP
#graphs	15,535	11,355
task	graph cls. (10 tasks)	node cls. (21 cls)
metric	average precision (AP)	macro F1
avg. nodes/graph	150.9	479.4
node features	9 (categorical)	14 (continuous)
edge features	3 (bond; water-level)	1 (boundary, used)
split	random 80/10/10, seed 0	official
modules/graph (mean)	127.1	planar, cycle-rich
module-tree depth (mean)	52.1	12.5 (et80)
non-singleton modules	3.1	more and larger
largest module	171	—

B.2 Architecture and training

All compared methods share: AtomEncoder (peptides) or linear encoder (SP), LayerNorm (unless stated), 4 propagation layers, hidden size 256, mean pooling over valid nodes (graph tasks), LayerNorm + linear head, AdamW (lr 10^{-3}, weight decay 0), ReduceLROnPlateau (factor 0.5, patience 20, min lr 10^{-5}), gradient clipping at norm 1, 250 epochs, batch size 128 (peptides bs128 protocol) or 64 (bs64 protocol) or 32 (SP). Parameter count $\approx$ 0.31M (mpnn, eq) and 0.45M (eq_adaptive, eq_modular) for the peptides models (within the LRGB $\sim$500k budget).

Solver configurations. *Global (eq, eq_adaptive)*: damped forward iteration, max 15 steps, tol 10^{-5}, damping 0.7; IFT backward, max 10 steps, tol 10^{-5}, damping 0.7. *Modular (eq_modular)*: S=6 sweeps, K=5 intra iterations, damping 0.7, unrolled exact gradients, optional sweep-level gradient checkpointing (used for K=8 at bs128). Topology (eq_adaptive and eq_modular): $A = 0.5A_{\text{given}} + 0.5A_{\text{QK}}$, A_{QK} per-row top-5 softmax over $QK^\top$ scores excluding existing edges/self/padding, row sums capped at 0.85 after mixing (0.95 for the bond part). Hardware: single RTX 4090; per-epoch times 17.5 s (eq_adaptive), 39.6 s (eq_modular, bs128), 12.5 s (eq), 6.7 s (mpnn).

B.3 Split details

The peptides main table uses a fixed random 80/10/10 permutation (seed 0) shared verbatim by all compared methods and seeds. SP always uses the official split shipped with the dataset.

C Additional Results

C.1 Per-seed main results (peptides-func, bs128)

Table 4 reports the per-seed numbers behind the peptides-func rows of Table 1.

Table 4: peptides-func test AP per seed.

Method	s42	s43	s44	mean±std
eq	48.20	48.07	48.49	48.25 ± 0.22
mpnn	53.93	54.37	54.76	54.35 ± 0.42
eq_adaptive	57.21	54.37	57.27	56.28 ± 1.66
eq_modular	58.13	55.72	55.78	56.54 ± 1.37

C.2 Tuning map (peptides-func, seed 42)

Both damping directions hurt, so 0.7 is a local optimum; sweeps beyond 6 give nothing. Intra iterations help at bs64: K=8 reaches 57.94 ± 1.59 over three seeds (59.61/57.75/56.45, T6), the best point on the budget–accuracy frontier; at bs128 the gain does not replicate even with checkpointing (53.48/55.19 for two seeds), so the main table reports the default budget (Table 1).

Table 5: Solver hyperparameter sweep around the default (S=6, K=5, damping 0.7, no block dropout). bs128 unless noted.

Run	Change	best val	test AP	note
main	—	59.22	58.13	default
T1	block dropout 0.1	57.96	56.15	hurt
T2	sweeps 10 (bs64)	59.33	57.22	saturated
T3	sweeps 3, K 3 (bs64)	58.68	56.64	$\frac{1}{3}$ budget, small loss
T4	damping 0.9	57.74	55.99	hurt
T5	damping 0.5	57.78	55.02	hurt
T6	K 8 (bs64)	60.13	59.61	best; bs128 requires checkpointing

C.3 Efficiency

Table 6: Wall-clock cost per epoch (single RTX 4090, 250-epoch runs) and parameter counts. Modular solving adds *no parameters* over the global solver (identical count); the cost is time only, and it is fixed in advance—the budget does not depend on the input.

Method	peptides-func (bs128)	PascalVOC-SP (bs32)	params
mpnn	6.7 s	—	0.31M
eq	12.5 s	—	0.31M
eq_adaptive (global)	17.5 s	15.9 s	0.45M (pep.) / 0.41M (SP)
eq_modular (ours)	39.6 s	32.4 s	0.45M (pep.) / 0.41M (SP)

Modular propagation costs 2.0–2.3× the wall time of global solving per epoch at identical parameter count, in exchange for exact unrolled gradients and a bounded, input-independent budget. Two offsets are worth noting: module decomposition is a one-time, deterministic preprocessing step (6 s for all 15,535 peptides), and activation memory—linear in the budget $S(K+1)$ by construction—can be traded back to $O(S)$ states with sweep-level checkpointing at ∼30% extra compute. Inference cost is fixed and known a priori, unlike tolerance-based solvers whose iteration count varies per input.

C.4 Negative results

We report these as found, since they constrain the method's story: (i) *Module dropout hurts everywhere we tried it*: ∼2 points on peptides-func (uniform p=0.1, T1), −3.0 F1 on SP (uniform p=0.1), and −1.8 F1 on SP with size-weighted rates (p=0.05, drop probability inversely proportional to module size, capped at 0.2). Protecting large modules reduces but does not eliminate the cost—modules are load-bearing (Section 6). (ii) *Weak coupling reverses the ordering*: at r=0.90 the global solver beats modular propagation (57.97 ± 1.23 vs. 54.68 ± 1.28, 3 seeds)—when fifteen damped iterations are effectively converged, modularization has nothing to offer. The method's regime is strong coupling. (iii) *Virtual nodes* hurt both arms (full analysis below). (iv) *Damping* 0.5 and 0.9 both hurt (T4, T5). (v) *Module importance does not correlate with aromaticity* ($\rho = -0.055$ over 200 test graphs), only with module size ($\rho = 0.31$). (vi) An older pilot comparing wall-clock speed across different `num_workers` settings suggested a 3× speed advantage of modular solving; this was a data-loading artifact. At matched settings, modular propagation costs 2.3× the wall time of global solving per epoch at bs128 (39.6 s vs. 17.5 s; Table in Appendix C.3), in exchange for exact gradients and bounded, input-independent budgets.

Virtual-node analysis (full). Adding a hub node connected to all nodes—the virtual node, worth +9 AP for GCN on LRGB—degrades both of our arms (−0.9 AP for eq_adaptive, −3.3 for eq_modular relative to matched baselines). Mechanistically: (a) the row-sum cap that keeps the fixed-point map contractive dilutes all bond weights once a degree-N hub is added, weakening the given graph's contribution; (b) a broadcast hub provides global exchange that the learned content-routed QK edges already supply, interfering with the learned routing; (c) for modular propagation specifically, the hub is a singleton module through which all inter-module traffic is funneled, homogenizing module states and eroding the modules' identity as distinct computational units. The

two long-range mechanisms—broadcast hub and learned routing—are thus mutually exclusive in our framework, which also explains most of our remaining gap to leaderboard numbers (Table 1).

C.5 PascalVOC-SP details

Table 7: SP per-seed test F1 (official split, edge weights, BatchNorm, dropout 0.1).

Method	s42	s43	s44	mean±std
eq_adaptive	25.67	26.38	25.94	26.00 ± 0.36
eq_modular (raw)	25.82	25.07	28.73	26.54 ± 1.93
eq_modular (et80)	27.24	28.13	26.28	27.21 ± 0.93
et80 + bd uniform 0.1	22.58	26.36	23.79	24.24 ± 1.93
et80 + bd size-weighted 0.05	25.55	25.82	24.85	25.41 ± 0.50

Dropout details for eq_modular: feature dropout 0.1 is essential (s42: without it, 21.33). Module-level (block) dropout hurts in every configuration tested (table above): uniform p=0.1 costs 3.0 F1, and size-weighted dropping (p=0.05, per-module probability inversely proportional to module size, capped at 0.2) still costs 1.8. An earlier observation that block dropout was roughly free came from the degenerate raw partition, where "dropping a module" meant freezing most of the graph; it does not replicate once the partition is meaningful. Without edge weights and BatchNorm (our initial, incorrect protocol), all methods sit at 12–14 F1; edge weights are the dominant factor on this dataset. Dropping all edges (the no-graph control quoted in Section 5) collapses the model to chance (3.9 F1).

C.6 Hard-regime per-seed numbers (bs64)

Table 8: Test AP at max row sum r.

r	Method	s42	s43	s44	mean±std
0.90	eq_adaptive	57.60	59.34	56.96	57.97 ± 1.23
0.92	eq_adaptive	57.23	54.25	56.15	55.88 ± 1.51
0.95	eq_adaptive	59.17	57.53	53.80	56.83 ± 2.78
0.98	eq_adaptive	54.93	53.55	57.31	55.26 ± 1.95
0.90	eq_modular	56.07	53.54	54.42	54.68 ± 1.28
0.92	eq_modular	56.87	59.59	57.96	58.14 ± 1.37
0.95	eq_modular	56.73	58.56	56.13	57.14 ± 1.24
0.98	eq_modular	55.61	58.67	55.39	56.56 ± 1.74

C.7 Water-level recipes

Do the solver differences survive at literature-standard recipes? We run both solvers at the official LRGB recipe (5 layers, hidden 300, 500 epochs) in three variants: plain (raw atom features), with standard boosters (bond-feature gating + 16-dim RWSE), and at doubled optimization steps (bs64, 250 epochs ≡ bs128, 500). Test AP / MAE, 3 seeds:

Table 9: peptides-func (AP ↑) and peptides-struct (MAE ↓) at water-level recipes. Boosters = bond gating + RWSE.

Recipe	Metric	mpnn	eq_adaptive	eq_modular
bs128, plain	AP	60.47 ± 0.29	$\mathbf{60.82 \pm 0.87}$	59.76 ± 0.81
bs128, boosters	AP	—	$\mathbf{64.11 \pm 0.37}$	63.76 ± 0.22
bs64, boosters	AP	—	$\mathbf{64.03 \pm 0.65}$	62.83 ± 0.47
bs128, plain, struct	MAE	$\mathbf{0.2863} \pm 0.001$	0.2995 ± 0.007	0.2914 ± 0.004

Three observations. (i) At generous recipes the solvers are *matched*: adaptive is up to 1.2 ahead on func, modular is ahead on struct (0.2914 vs. 0.2995 MAE). (ii) The shrinkage of the solver gap

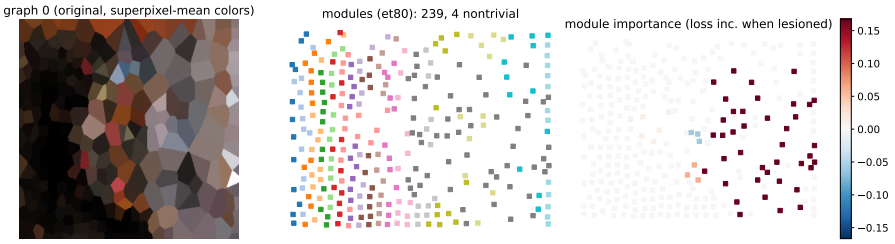


Figure 6: Additional module-attribution example on PascalVOC-SP (same format as Figure 2).

tracks the theory: boosters supply structural information (positional encodings, bond features) that otherwise must come from the cyclic modules, so the solver's organization matters less. (iii) The ablation tells the same story in reverse: at this recipe a *random partition* of the same size distribution costs nothing (64.22 ± 0.89 vs. 63.76 ± 0.22 for the bridge decomposition)—versus -4.8 points under our feature-poor main recipe (Table 2). The value of the cyclic structure is proportional to how much the input features depend on it.

C.8 Module-tree depth and sweep coverage

How far does one layer propagate information along the module structure? We measure the depth of the bridge-block tree (longest inter-module path, in module-hops): peptides-func averages depth 52.1 (median 47, p90 92), PascalVOC-SP under et80 averages 12.5 (median 11). A single layer with S=6 sweeps therefore does *not* cover the tree. Three factors close the gap. (i) Layers compound: L layers give $S{\cdot}L$ hops of total reach. (ii) The learned QK edges are content-routed shortcuts across the module tree, so the solver's effective reach exceeds the tree analysis. (iii) Empirically the budget saturates at S=6 (Figure 4a)—full-depth coverage is not what the task rewards, consistent with the nominal-equilibrium finding (Section 6): the extra long-range hops change the fixed point but not the label.

D Extended Discussion and Limitations

Modular equilibrium reframes fixed-point graph computation as a structured, bounded, and analyzable process: equilibrium where feedback lives, feed-forward where it does not. At matched topology it matches or beats global solving, is more robust under strong coupling, and is uniquely analyzable at module granularity (lesion, attribution).

Limitations. (i) On easy regimes (small graphs, weak coupling) the advantage vanishes and can reverse—at r=0.90 the global solver wins by 3.3 AP (Appendix C.4). The method's motivation is structural, and we do not claim superiority there. (ii) Absolute numbers sit below heavily tuned leaderboard entries; the gap decomposes into protocol choices and mutually exclusive components (Section C.4), and combining modular solving with stronger backbones is future work. (iii) Unrolled gradients trade memory for fidelity; sweep-level checkpointing mitigates but does not eliminate the cost. (iv) The budget (S, K) is a hyperparameter, though the method is insensitive around the default (Figure 4a). (v) Module-level dropout is a diagnostic probe, not a usable regularizer (Appendix C.4).

E Analysis Protocols

Residual measurement. For each equilibrium layer, we recompute the layer input by replaying the model up to that layer on test batches (5 batches, ≈640 graphs), rebuild the layer's A_{mix} from its own learned projections, and run (i) the trained-budget iteration (15 steps, damping 0.7), recording steps used and the final residual $\|z - \sigma(u + Az)\|_\infty$; and (ii) a continuation to tol 10^{-6} (max 500 steps) to count the missing iterations. For eq_modular we additionally report the intra-module residual after the last sweep and the distance to the true global fixed point solved to 10^{-6}.

Converged evaluation. The "true convergence" number of Section 6 evaluates the trained eq_adaptive checkpoint with the forward solver forced to 300 iterations, tol 10^{-7}.

Lesion study. On the trained eq_modular checkpoint (peptides-func, seed 42), we freeze all nodes of k randomly sampled non-singleton modules at their initial states in every modular layer and re-evaluate test AP. The node-lesion control is *count-matched*: for each sampled set of modules, we freeze the same number of randomly sampled valid nodes, so both conditions remove identical node counts. $k \in \{1, 2, 4, 8\}$; sampling is seeded and identical between conditions. Matched-count test AP: intact 58.13; module lesions $55.29/54.54/51.93/51.19$ for $k{=}1/2/4/8$; matched node lesions $56.83/55.67/55.17/54.05$.

Module attribution. Over the first 200 test graphs we measure, per non-singleton module, the increase in BCE loss when that module is lesioned (all layers), and record module size and aromatic-atom fraction. We report mean/median loss deltas and Spearman-style correlations (size 0.306, aromaticity -0.055).

Additional visualization. A second module-attribution example on PascalVOC-SP (graph 0, Figure 6): the high-importance module covers the foreground object, consistent with Figure 2.