GRACE: A MECHANISTIC UNDERSTANDING OF GNNS THROUGH CIRCUIT DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Mechanistic interpretability aims to reverse-engineer neural networks by identifying the circuits that implement specific behaviors. Despite rapid progress in language models, no framework for mechanistic circuit discovery exists for Graph Neural Networks (GNNs). We argue that the obstacle is not the circuit-finding machinery, which transfers from language models, but the steps that precede it: in graphs, behaviors are neither known nor readily hypothesized, ground truth is unavailable, and discrete edits cannot corrupt a behavior without breaking the graph. We propose *Graph Circuit Explorer* (GRACE), which discovers behaviors without supervision by clustering graphs in the model's functional embedding space and characterizing each cluster by a subgraph signature, selected with the best approximation guarantee achievable in polynomial time. GRACE corrupts each behavior by continuously dampening its signature, which preserves graph validity and enables gradient-based attribution patching, and extracts circuits under an edge budget, a problem we prove NP-hard. Across six datasets and four architectures, GRACE recovers circuits that are sufficient and necessary at a small fraction of the model's edges and outperform size-matched random circuits in every setting. On MUTAGENICITY, the discovered behaviors correspond to known toxicophores, and circuit analysis reveals that behaviors share most of the hidden units they use, structure that is invisible to input-level explanations.

1 INTRODUCTION AND RELATED WORK

Explainability is undergoing a fundamental change. Just as psychology treated the mind as a black box due to a lack of introspective tools, XAI has predominantly relied on black-box techniques analyzing models based on input-output relationships neglecting the model's internal architecture (Longo et al., 2024; Cheng et al., 2025; Lyu et al., 2024; Zhang et al., 2024). However, the field is moving towards a more granular approach focusing on the internal mechanics of deep neural networks (Bereska & Gavves, 2024; Rai et al., 2024). GNN explainability has not kept pace with this shift. The prevailing research centers on identifying the decision-critical substructures of an input graph, which may be instance-specific (local explanations) (Zhang et al., 2024; Longo et al., 2024; Kosan et al., 2024) or shared across instances (global explanations) (Armgaan et al., 2025; Fournier & Medya, 2025; Armgaan et al., 2024; Wang et al., 2024). To our knowledge, no framework for mechanistic circuit discovery in GNNs exists. A detailed discussion on GNN explainers is in App. B.

Mechanistic interpretability (MI) aims to completely specify a neural network's computation through its components (Bereska & Gavves, 2024; Rai et al., 2024). Early work assumed that individual neurons encode individual features, but neurons proved largely *polysemantic* (Olah et al., 2020; Marshall & Kirchner, 2024), and features came to be modeled as directions in activation space (Nanda et al., 2023; Nanda, 2023; Huben et al., 2024; Whittington et al., 2023), despite some criticism (Engels et al., 2025; Li et al., 2023). *Circuits* are the computational counterpart: subgraphs of the network's computational graph that pass information between features across layers to implement a behavior (Olah et al., 2020). Among circuit discovery algorithms (Conmy et al., 2023; Syed et al., 2024; Hanna et al., 2024; He et al., 2025), attribution patching (Syed et al., 2024; Hanna et al., 2024) is one of the prominent approaches. It treats a neural network as a causal graph and performs interventions to isolate the components causally responsible for a behavior. The standard protocol proceeds as follows: (1) hypothesizing that a behavior exists and characterizing it; (2) isolating the behavior and constructing a dataset that targets it specifically; (3) faithfully corrupting the behavior

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.

without disturbing any other behaviors the model may have acquired; and (4) patching activations from a corrupted input into the clean run to identify the responsible circuit.[1]

Roadblock 1: identifying behaviors. A mechanistic analysis of GNNs fails at the very first step. In LLMs, behaviors are either well-known or readily hypothesized, e.g., subject-verb agreement, indirect object identification, and factual recall. However, an analogous behavioral vocabulary does not exist for GNNs. One can quantify that a GNN successfully distinguishes between toxic and non-toxic molecular graphs, but the internal behaviors it exploits to do so remain opaque. This opacity is compounded by *the unavailability of ground truth*, a known issue in graph XAI. In graph ML, unlike in vision or NLP, the ground truth for which features or substructures determine class membership is unknown, so there is no reference point against which to interpret what the model has learned.

Roadblock 2: corrupting behaviors. Even granting that a behavior can be identified, corrupting it without disturbing the graph structure, and thereby inadvertently perturbing other behaviors, is not straightforward. In LLMs, minimal counterfactual pairs can be handcrafted with surgical precision: to corrupt subject-verb agreement, one contrasts "The cat *sits*" with "The cat *sit*"; to corrupt factual recall, one substitutes "Rome is the capital of Italy" with "London is the capital of Italy". Behavior corruption is tricky in graphs as a perturbation propagates through the graph in complex and uncontrollable ways, and discrete edits change the graph's size or validity, breaking the correspondence between clean and corrupted activations that patching relies on.

Contributions. To overcome the above-mentioned roadblock and enable mechanistic circuit discovery for GNNs, we propose *GRaph Circuit Explorer* (GRACE). Our contributions are as follows.

- **Problem formulation.** We formulate the problem of mechanistic circuit discovery in GNNs.
- **Methodology.** GRACE discovers behaviors without supervision by clustering graphs in the GNN's functional embedding space and characterizing each cluster by a subgraph *signature*. The signature doubles as the handle for corruption: dampening it continuously preserves graph validity and yields the interpolation path that EAP-IG requires.
- **Theory.** Both combinatorial steps of GRACE are NP-hard. Greedy signature selection attains the best approximation achievable in polynomial time $(1 - 1/e)$, and our hardness result for budgeted circuit extraction resolves a question left open by **?** and holds for any architecture.
- **Experiments.** Across six datasets and four architectures, GRACE's circuits are sufficient and necessary at a small fraction of the model's edges and beat size-matched random circuits in all 24 settings. On MUTAGENICITY, the behaviors match known toxicophores, and circuits reveal that behaviors share most of their hidden units, structure invisible from the input side.

2 PRELIMINARIES AND PROBLEM FORMULATION

2.1 GRAPH CLASSIFICATION WITH GNNS

We consider the supervised graph classification task. Let $\mathcal{G} = \{G_1, \cdots, G_N\}$ denote a dataset of N graphs, where each graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$ consists of nodes $\mathcal{V}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, node features $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, and edge features $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d'}$ with d, d' being the feature dimensions. Each graph is associated with a class label $y_G \in \{1, \ldots, c\}$. A GNN learns a function $\Phi : \mathcal{G} \to \mathbb{R}^c$ that maps graphs to class logits through iterative message passing. At layer ℓ, node representations are updated via:

$$\mathbf{h}_v^{(\ell+1)} = \texttt{Update}^{(\ell)}\left(\mathbf{h}_v^{(\ell)}, \texttt{Aggregate}^{(\ell)}\left(\{\mathbf{h}_u^{(\ell)} : u \in \mathcal{N}(v)\}\right)\right), \tag{1}$$

where $\mathcal{N}(v)$ denotes the neighborhood of node v, and `Update` and `Aggregate` are learnable functions. After L layers, node representations are pooled to obtain a graph-level representation $\mathbf{h}_G = \texttt{Readout}(\{\mathbf{h}_v^{(L)} : v \in \mathcal{V}\})$, which is then passed to a classifier to produce logits.

2.2 CIRCUIT DISCOVERY

Definition 1 (Behavior) *A behavior B is a pattern present in a subset $\mathcal{S} \subset \mathcal{G}$ of input graphs, which is identified and employed by the GNN Φ in its downstream task. B qualifies as a behavior iff:*

(i) ***Locality.*** *There exists a proper subset $\mathcal{S} \subset \mathcal{G}$, with $\mathcal{S} \neq \emptyset$ and $\mathcal{S} \neq \mathcal{G}$, such that B is present in $G \in \mathcal{S}$ and absent from $\mathcal{G} \setminus \mathcal{S}$.*

[1] See App. D for a detailed explanation.

(ii) ***Relevance.*** *The pattern B is utilized by the* GNN Φ *in its downstream task.*

Intuitively, a behavior partitions the input space into graphs that exhibit it and graphs that do not, and this partition must be meaningful to Φ: the model must actively exploit the pattern when computing its output. This second condition distinguishes a behavior from an incidental regularity. A graph dataset may contain patterns that the model disregards in favor of more predictive signals. Such patterns do not constitute behaviors. The definition anchors behaviors to the model's computation rather than to statistical regularities in the data.

Definition 2 (Computation Graph) *Let Φ be a trained* GNN. *Its computation graph $\Phi_C = (V, E)$ is a directed acyclic graph (DAG) in which nodes represent neurons and directed edges represent the flow of information between them. The activation of an edge $e = (u, v) \in E$ is the signal transmitted from neuron u to neuron v during a forward pass.*

The definition of a neuron depends on the chosen granularity of analysis: an entire MLP block, an individual layer, an attention head, or a single row $W[i]$ of a weight matrix W. In this work, we adopt the finest granularity, treating each row as a neuron, such that a weight matrix $W \in \mathbb{R}^{p \times q}$ has p neurons. An edge is then a single entry $W[i, j]$ of that matrix, carrying the signal from the j-th neuron of the preceding layer to the i-th neuron of this one.

Definition 3 (Circuit) *Let $\Phi_C = (V, E)$ be the computation graph of the* GNN Φ*, where V is the set of neurons and E is the set of directed edges connecting them. A circuit $C = (V_C, E_C)$ is a subgraph of Φ_C, with $V_C \subseteq V$ and $E_C \subseteq E$, that suffices to implement a behavior B and is* connected end to end*: every neuron in V_C lies on a path within C that runs from an input neuron to an output neuron. Equivalently, C has no leaf neurons other than in the input and output layers, i.e., every neuron in between retains at least one parent and at least one child within C. Behaviors and circuits are in correspondence $B_i \mapsto C_i$: for every behavior B_i identified and utilized by Φ, there exists a circuit $C_i \subseteq \Phi_C$ that implements it.*

A circuit is the computational substrate of a behavior. While a behavior is defined over the input space, its corresponding circuit is defined over the model's internals, as the subgraph of Φ_C that is causally responsible for that behavior.

Problem 1 *Given a trained* GNN Φ *and its computation graph Φ_C,*

(i) ***Behavior identification & isolation.*** *Identify the complete set of behaviors $\mathcal{B} = \{B_1, \ldots, B_k\}$ that Φ exploits in solving its downstream task, along with the corresponding partition of $\mathcal{G}$ into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that each B_i is present in $\mathcal{S}_i$ and absent from $\mathcal{G} \setminus \mathcal{S}_i$.*

(ii) ***Circuit localization.*** *For each identified behavior $B_i \in \mathcal{B}$, recover the circuit $C_i \subseteq \Phi_C$ that implements it, i.e., $B_i \mapsto C_i$ and C_i is sufficient to reproduce B_i.*

3 GRACE: PROPOSED METHODOLOGY

Fig. 1 presents the 3-phase circuit discovery pipeline of GRACE. In the first phase, we isolate behavioral characteristics that have been learned by the GNN. In phase two, we corrupt these behaviors and diagnose the impact on the computation graph through activation patching. Finally, in phase 3, we extract the circuits that are responsible the for the GNN's predictions.

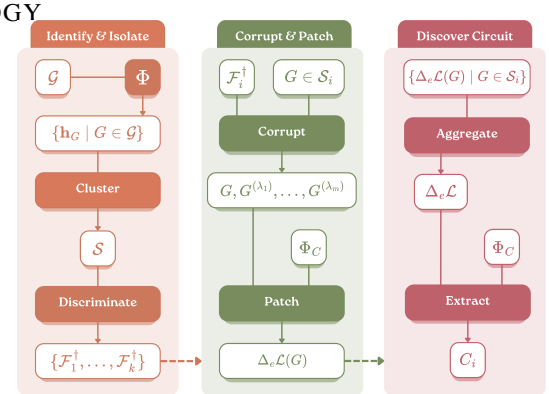


Figure 1: The 3-phase circuit discovery pipeline of GRACE.

3.1 ROADBLOCK 1: BEHAVIOR IDENTIFICATION & ISOLATION

Recall from § 1 that, unlike in LLMs, where a rich behavioral vocabulary exists, the behaviors a GNN acquires to solve a task are opaque to the practitioner. We address this challenge through an unsupervised approach that delegates behavior identification to the model itself.

Clustering in functional embedding space. For each predicted class $\hat{y} \in \{1, \ldots, c\}$, we collect the set of graphs $\{G \in \mathcal{G} : \Phi(G) = \hat{y}\}$. We operate over Φ's predictions rather than ground-truth labels,

since our objective is to characterize the model's own decision-making strategy. For each such graph G, we extract the graph-level embedding $\mathbf{h}_G$ from Φ, and apply a clustering algorithm to the resulting set of embeddings. While we use k-means, other clustering algorithms may also be used.

The key property enabling this approach is that GNNs implicitly learn to respect *functional* rather than structural distance. Negishi et al. (2025) demonstrate that Message-Passing Neural Networks (MPNNs) are trained to capture task-relevant functional differences between graphs rather than superficial differences in graph topology. They further find that MPNNs focus on compact, functionally important sub-regions within graphs to capture this functional distance. This result has two important consequences for our setting: First, proximity in the embedding space $\mathbf{h}_G$ reflects functional similarity as perceived by the GNN Φ. Hence, clusters in this space are defined by the model's own internal geometry rather than by an external criterion. Second, since the embedding space encodes task-relevant functional distance, membership in a cluster is inherently tied to the model's downstream computation, guaranteeing that patterns shared within a cluster are ones the model actively relies upon, thus satisfying the relevance condition of Definition 1.

From clusters to behaviors. We hypothesize that each cluster $\mathcal{S}_i$ in the functional embedding space corresponds to a distinct behavior B_i. Under this hypothesis, the locality condition of Definition 1 is satisfied by construction: B_i is present in every $G \in \mathcal{S}_i$ and absent from $\mathcal{G} \setminus \mathcal{S}_i$. The relevance condition is satisfied by the functional distance argument above. In particular, this formulation is unsupervised: the practitioner is not required to hypothesize, name, or characterize any behaviors in advance. The GNN itself induces the partition, sidestepping the opacity issue identified in § 1.

Mining behavior signatures. The partition $\mathcal{S}_i$ tells us *which* graphs share a behavior, but not *which patterns* define it. Subgraph patterns are a natural vocabulary for this characterization. Message passing computes each node representation from the node's L-hop neighborhood (Eq. 1). Consequently, the evidence a GNN aggregates into $\mathbf{h}_G$ is carried by local substructures. Motivated by this observation, we wish to characterize each cluster $\mathcal{S}_i$ by a small set of subgraph patterns that are present in its graphs and absent from the rest. We formally set up this problem as follows.

Let $\mathcal{D}$ denote the set of all connected subgraph patterns; we discuss the exact process of constructing $\mathcal{D}$ at the end of this section. For $f \in \mathcal{D}$, we use the notation $f \sqsubseteq G$ to denote f is subgraph isomorphic to G. A pattern f *discriminates* a pair (G_a, G_b) if $G_a \in \mathcal{S}_i,\ G_b \in \mathcal{G} \setminus \mathcal{S}_i,\ f \sqsubseteq G_a$ and $f \not\sqsubseteq G_b$, i.e., f marks G_a as a member of the cluster and G_b as a non-member. The set of all such pairs that f successfully discriminates is $R(f) = \{(G_a, G_b) \in \mathcal{S}_i \times \mathcal{G} \setminus \mathcal{S}_i : f \sqsubseteq G_a,\ f \not\sqsubseteq G_b\}$.

Since a cluster may be marked by several substructures (§3.1), a single pattern rarely discriminates every pair. Hence, we introduce the concept of *discrimination coefficient* $q(\mathcal{F})$ of a pattern set $\mathcal{F} \subseteq \mathcal{D}$ as the fraction of pairs the set $\mathcal{F}$ successfully discriminates:

$$q(\mathcal{F}) = \frac{\left|\bigcup_{f \in \mathcal{F}} R(f)\right|}{|\mathcal{S}_i||\mathcal{G} \setminus \mathcal{S}_i|} \in [0, 1] \tag{2}$$

The optimal vocabulary within a budget s, that characterizes the behavior of cluster $\mathcal{S}_i$ can now be mapped to the following optimization problem.

Problem 2 (Behavior Signatures) *Given a cluster of graph $\mathcal{S}_i$, a candidate pool $\mathcal{D}$, and a budget s, find $\mathcal{F}^* = \arg\max_{\mathcal{F} \subseteq \mathcal{D},\, |\mathcal{F}| \leq s} q(\mathcal{F})$.*

The budget s keeps the signature compact and bounds the portion of each graph that the corruption operator touches.

Proposition 1 *Problem 2 is NP-hard, and no polynomial-time algorithm approximates it within a factor better than $1 - 1/e$ unless $\mathrm{P} = \mathrm{NP}$.*

PROOF. See App. C.2.

Fortunately, our problem is monotonic and submodular, and thus admitting an $1-1/e$ approximation through the greedy forward selection algorithm.

Proposition 2 *The coefficient q is monotone and submodular with $q(\emptyset) = 0$. Hence, greedy forward selection (Alg. 1) returns $\mathcal{F}^\dagger$ with $q(\mathcal{F}^\dagger) \geq (1 - 1/e)\, q(\mathcal{F}^*)$.*

PROOF. See App. C.1.

Greedy selection starts from $\mathcal{F}_0 = \emptyset$ and, at each step t, adds the pattern that resolves the most pairs not yet resolved:

$$f_t = \underset{f \in \mathcal{D} \setminus \mathcal{F}_{t-1}}{\arg\max}\ q\big(\mathcal{F}_{t-1} \cup \{f\}\big) - q\big(\mathcal{F}_{t-1}\big), \qquad \mathcal{F}_t = \mathcal{F}_{t-1} \cup \{f_t\}. \tag{3}$$

It stops after s steps, or earlier if no pattern resolves a new pair, and returns the final $\mathcal{F}_t$ as the signature $\mathcal{F}_i^\dagger$. Signatures are extracted once per cluster. Clustering is performed separately within each predicted class $\hat{y} \in \{1, \ldots, c\}$ (§3.1), and we index the clusters of all classes jointly as $\mathcal{S}_1, \ldots, \mathcal{S}_k$. The behavior B_i is then operationalized as the pattern set $\mathcal{F}_i^\dagger$ taken as a whole. A graph in $\mathcal{S}_i$ is not required to contain every member of $\mathcal{F}_i^\dagger$: it expresses B_i through whichever members it does contain, and the corruption operator of §3.2 acts on all of their occurrences.

Constructing the candidate pool $\mathcal{D}$. We take $\mathcal{D}$ to be the frequent connected subgraphs of $\mathcal{S}_i$, obtained with a frequent subgraph miner (Kuramochi & Karypis, 2004; Yan & Han, 2002; Nijssen & Kok, 2005). One may also use subgraph explanations provided by global GNN explainers (Azzolin et al., 2023; Armgaan et al., 2024; Wang et al., 2024) albeit they are expensive and require training.

3.2 Roadblock 2: Behavior Corruption

Having identified the behavior set $\mathcal{B} = \{B_1, \ldots, B_k\}$ and their corresponding graph partitions, we now address the second roadblock raised in § 1: how to faithfully corrupt a behavior B_i without inadvertently disturbing the rest of the graph or the other behaviors GNN Φ may have acquired.

The inadequacy of discrete perturbations. The immediate corruption strategy is to remove the discriminative subgraph corresponding to B_i, i.e., $\mathcal{F}_i^\dagger$, entirely. This is, however, untenable for the following reasons. First, removing nodes changes the graph's size, breaking the correspondence between activations in the clean and corrupted forward passes that patching requires. Second, the structural excision is too drastic: the nodes neighboring the removed subgraph lose their edges abruptly, inducing an uncontrolled cascade of changes in their aggregated messages. Third, in many domains, such as molecular graphs, removing nodes produces chemically invalid structures, undermining the integrity of the intervention.

Continuous message dampening. To negate the collateral damage of discrete perturbations, we adopt a softer variant through *continuous dampening*.

For each pattern $f \in \mathcal{F}_i^\dagger$, we locate its instances in G via subgraph isomorphism, yielding a matched set of nodes $\mathcal{V}_f(G) \subseteq \mathcal{V}$ and edges $\mathcal{E}_f(G) \subseteq \mathcal{E}$. Rather than masking the edges in $\mathcal{E}_i(G)$, we downweigh them by a dampening factor $\lambda \in [0, 1)$. We collect the nodes and edges to be dampened as $\mathcal{V}_i(G) = \bigcup_{f \in \mathcal{F}_i^\dagger} \mathcal{V}_f(G)$ and $\mathcal{E}_i(G) = \bigcup_{f \in \mathcal{F}_i^\dagger} \mathcal{E}_f(G)$; so nodes or edges that participate in multiple isomorphisms are not dampened multiple times. For each edge $(u, v) \in \mathcal{E}_i(G)$, the message passed from u to v during aggregation is scaled by λ:

$$\mathbf{m}_{u \to v}^{(\ell)} \leftarrow \lambda \cdot \mathbf{m}_{u \to v}^{(\ell)}, \quad \forall\, (u, v) \in \mathcal{E}_i(G),\ \forall\, \ell \in \{1, \ldots, L\}.$$

Node feature dampening. Since graph classification involves a global pooling step $\mathbf{h}_G = \text{POOL}(\{\mathbf{h}_v^{(L)} : v \in \mathcal{V}\})$, which aggregates node representations directly without passing through any edge, we additionally downweight the input node features of all nodes $v \in \mathcal{V}_i(G)$:

$$\mathbf{x}_v \leftarrow \lambda \cdot \mathbf{x}_v, \quad \forall\, v \in \mathcal{V}_i(G). \tag{4}$$

Together, edge and node feature dampening constitute our corruption operator. For a graph $G \in \mathcal{S}_i$, the corrupted graph G^- is obtained by applying both interventions with factor λ to the nodes $\mathcal{V}_i(G)$ and edges $\mathcal{E}_i(G)$ identified within G. The pair (G, G^-) forms the clean-corrupt pair required for the patching protocol described in § 3.3.

3.3 Patching

With the behavior set $\mathcal{B} = \{B_1, \ldots, B_k\}$ and the corresponding clean-corrupt pairs $\{(G, G^-) \mid G \in \mathcal{S}_i\}$ in hand, we now recover the circuit $C_i \subseteq \Phi_C$ responsible for each behavior B_i.

Activation patching at edge granularity. Patching treats Φ_C as a causal graph and asks, of each edge in turn, how much of the model's response to B_i travels through it. Let $\mathbf{a}_e(G)$ denote the activation carried by edge $e = (u, v) \in E$ when Φ is run on G, and let $\mathcal{L}$ denote a task metric that quantifies how far a forward pass has drifted from the clean one. A single patch substitutes $\mathbf{a}_e(G^-)$ for $\mathbf{a}_e(G)$ while every other edge retains its clean activation; the resulting movement in $\mathcal{L}$ is the causal contribution of e.

Performed exhaustively, this is the ground truth we would like to have, but it costs one forward pass per edge. This is a scalability bottleneck since $|E|$ runs into the thousands even for compact GNNs, and the sweep must be repeated for every graph in every behavior cluster. Exhaustive patching is therefore out of reach, hence we use the approximation strategy of EAP-IG (Hanna et al., 2024).

Hanna et al. (2024) averages the gradient along a path from the clean input to the corrupted one. Writing $G^{(\lambda_k)}$ for the graph corrupted at intermediate strength λ_k, with

$$\lambda_k = 1 - \frac{k}{m}(1-\lambda), \qquad k = 1, \ldots, m, \tag{5}$$

interpolating between the clean graph at $\lambda_k = 1$ and the fully corrupted G^- at $\lambda_k = \lambda$, the score becomes:

$$\Delta_e \mathcal{L}(G) = \big(\mathbf{a}_e(G^-) - \mathbf{a}_e(G)\big)^\top \frac{1}{m} \sum_{k=1}^{m} \frac{\partial \mathcal{L}}{\partial \mathbf{a}_e}\bigg|_{\mathbf{a}_e(G^{(\lambda_k)})}. \tag{6}$$

The activation difference is unchanged; only the gradient is now a path average, which gives every stretch of the corruption a say in the score rather than the clean endpoint alone. Theis requires $m+2$ forward passes and m backward passes per graph, with m being a small constant.

The task metric $\mathcal{L}$. We take $\mathcal{L}$ to be the Kullback-Leibler divergence of the logits of patched output from that of the clean one,

$$\mathcal{L}\big(\Phi^{\text{patched}}\big) = D_{\text{KL}}\big(\Phi(G) \,\|\, \Phi^{\text{patched}}(G)\big), \tag{7}$$

with the clean model as the reference.

From a graph to a behavior. Eq. 6 scores an edge on a single clean-corrupt pair, whereas a behavior is a property of the whole cluster. We therefore average the per-graph scores over $\{(G, G^-) \mid G \in \mathcal{S}_i\}$,

$$\Delta_e \mathcal{L} = \frac{1}{|\mathcal{S}_i|} \sum_{G \in \mathcal{S}_i} \Delta_e \mathcal{L}(G), \tag{8}$$

and rank edges by the magnitude of that average.

Circuit extraction. Scoring yields a weight on every edge of Φ_C; extraction must turn these into the circuit C_i. Recall from Def. 3 that a circuit must be a subgraph of Φ that is connected end-to-end and thereby allowing information flow from the first layer of the GNN to the last. Extraction is thus a constrained optimization over Φ_C, which we state formally.

Problem 3 (Heaviest circuit) *Let $\Phi_C(V, E, W)$ be the computation graph of Φ weighted by the attribution scores, $W = \{|\Delta_e \mathcal{L}|\}_{e \in E}$, and let k be an edge budget. Find the heaviest circuit*

$$C^* = \underset{\substack{C=(V_C, E_C) \subseteq \Phi_C \\ |E_C| \le k}}{\arg\max} \sum_{e \in E_C} |\Delta_e \mathcal{L}|,$$

where C is the set of all possible circuits of Φ.

Theorem 1 *Given a weighted computation graph $\Phi_C(V, E, W)$ and a budget k, it is NP-hard to find a heaviest circuit $C^*(V_{C^*}, E_{C^*}) \subseteq \Phi_C$ subject to $|E_{C^*}| \le k$.*

PROOF. See App. E.

Hanna et al. (2024) observe that a budgeted circuit search of this form is *likely* NP-hard, but leave the question open, which we formally prove by a reduction from Set Cover. As a result, an exact solution is unavailable and a heuristic is required. We choose greedy. App. F gives the procedure in full. In brief, the circuit is initialized at the output logits and grown backwards: at each step, the heaviest edge whose child already lies in the circuit is admitted. The greedy design of growing backwards guarantees from the logit layer guarantees that a childless neuron is never introduced, since the child of every admitted edge was already in the circuit when the edge was considered. The rule does not, however, guarantee that the traversal reaches the input features before the budget is exhausted. A circuit that remains unanchored at the input is hence discarded.

4 EXPERIMENTS

In this section, we benchmark GRACE on graph classification across six datasets (Table 1) and four GNN architectures, and evaluate the following research questions.

Table 1: Statistics of the datasets used to benchmark GRACE. Further details on the dataset semantics are discussed in App. G.

Dataset	#Graphs	Avg. #nodes	Avg. #edges	#Node feat.	#Edge feat.
cycles (Ying et al., 2019)	1,000	28.75	28.26	5	0
MUTAG (Debnath et al., 1991)	188	17.93	19.79	7	4
PROTEINS (Borgwardt et al., 2005)	1,113	39.06	72.82	3	0
NCI1 (Wale et al., 2008)	4,110	29.87	32.30	37	0
MUTAGENICITY (Kazius et al., 2005)	4,337	30.32	30.77	14	3
ogbg-molhiv (Hu et al., 2020)	41,127	25.51	27.47	14	4

- **RQ1: Faithfulness.** Are the recovered circuits sufficient and necessary for their behaviors, and at what fraction of the computation graph?
- **RQ2: Domain validity.** Where ground truth exists, do the discovered behaviors correspond to known domain knowledge?
- **RQ3: Circuit structure.** How do the circuits of different behaviors share the model's hidden units, and what does this imply for analyzing and editing GNNs?

At this juncture, we re-emphasize that existing GNN explainers, local (Ying et al., 2019) or global (Armgaan et al., 2024; 2025), identify *which* input substructures a prediction depends on. GRACE uses such substructures only as a handle: its output is the set of behaviors the model has learned and, for each, the circuit that computes it. Where GRACE's signature coincides with an explainer's subgraph, the two agree on what the model responds to; only the circuit reveals how and where the response is computed. Hence, *existing explainers do not serve as baselines.*

Our implementation is available at `https://anonymous.4open.science/r/grace-anonymous-1435/`. The experimental setup in terms of hardware and software environment, data splits, hyper-parameters, etc. are outlined in App. G.

4.1 CIRCUIT EVALUATION METRICS

Having recovered a circuit C_i for each behavior B_i, we quantify how faithfully C_i implements B_i through two complementary questions. Is C_i *sufficient*: does the model still respond as it does on the clean graph when only the edges of C_i carry clean activations? And is C_i *necessary*: does corrupting the edges of C_i alone reproduce the effect of corrupting the entire graph?

Both questions are answered by running Φ on a clean graph $G \in \mathcal{S}_i$ while some edges are patched, i.e., made to carry the activations they had on the corrupted graph G^-. The two scores differ only in which edges are patched. In the *circuit-only* run $\Phi_{C_i}(G)$, every edge outside C_i is patched, so the circuit is the only clean pathway to the logits. In the *circuit-knockout* run $\Phi_{\setminus C_i}(G)$, only the edges of C_i are patched, and the rest of the network is left clean. In both runs, unpatched edges are recomputed as usual, so the effect of the patched edges propagates downstream. Let $Z_i = \sum_{G \in \mathcal{S}_i} \mathcal{L}\big(\Phi(G^-)\big)$ be the total movement produced by full corruption, measured with the task metric $\mathcal{L}$ of Eq. 7. It is the largest movement the corruption of B_i can produce, and it serves as the denominator of both scores.

Sufficiency measures how close the output stays to the clean model when clean information reaches the logits only through C_i:

$$\sigma(C_i) = 1 - \frac{1}{Z_i} \sum_{G \in \mathcal{S}_i} \mathcal{L}\big(\Phi_{C_i}(G)\big). \tag{9}$$

Necessity measures how far corrupting C_i alone moves the output, relative to corrupting the whole graph:

$$\nu(C_i) = \frac{1}{Z_i} \sum_{G \in \mathcal{S}_i} \mathcal{L}\big(\Phi_{\setminus C_i}(G)\big). \tag{10}$$

The two runs recover the clean and corrupted passes at the extremes. For the empty circuit, the circuit-only run patches every edge and the knockout run patches none, so $\Phi_{\emptyset}(G) = \Phi(G^-)$ and $\Phi_{\setminus \emptyset}(G) = \Phi(G)$. For the full computation graph $C_i = \Phi_C$, the roles reverse. Both scores therefore equal 0 for the empty circuit and 1 for the full model, and a faithful circuit approaches 1 at a small fraction of $|\Phi_C|$.

4.2 RQ1: FAITHFULNESS

Circuit performance. Fig. 2 plots sufficiency and necessity against circuit size for one behavior per dataset in GCN. Sufficiency approaches 1 once the circuit holds at most 10% of the edges of Φ_C, whereas a size-matched random circuit stays near 0 until almost the whole model is retained.

Table 2: Advantage of GRACE over a size-matched random circuit expressed through AUC(GRACE) − AUC(random); higher is better, 0 means no advantage. Mean over the behaviors of each dataset–architecture pair, std in small type.

		Cycles	MUTAG	MUTAGENICITY	PROTEINS	NCI1	MOLHIV	Avg.
Sufficiency	GCN	0.31±0.01	0.48±0.08	0.47±0.15	0.45±0.05	0.36±0.02	0.58±0.17	0.44
	GAT	0.55±0.04	0.60±0.11	0.38±0.15	0.55±0.04	0.33±0.13	0.48±0.12	0.48
	GIN	0.37±0.02	0.51±0.02	0.31±0.05	0.56±0.06	0.21±0.07	0.36±0.08	0.39
	GRAPHSAGE	0.49±0.05	0.43±0.03	0.58±0.03	0.71±0.02	0.46±0.05	0.35±0.05	0.50
Necessity	GCN	0.39±0.01	0.50±0.03	0.39±0.12	0.46±0.08	0.38±0.13	0.50±0.11	0.44
	GAT	0.55±0.04	0.63±0.07	0.46±0.07	0.48±0.18	0.47±0.12	0.42±0.10	0.50
	GIN	0.55±0.01	0.54±0.04	0.49±0.07	0.69±0.08	0.50±0.09	0.43±0.07	0.53
	GRAPHSAGE	0.57±0.01	0.12±0.07	0.51±0.20	0.65±0.12	0.60±0.02	0.36±0.12	0.47

Necessity shows a similar separation: the circuit-knockout run moves the output about as far as full corruption does at a small fraction of the model's edges, while knocking out a random circuit of equal size has little effect until it approaches the full model.[2]

Fig. 2 shows sufficiency and necessity for the behavior corresponding to the largest cluster per dataset. To compare GRACE against the random control over *all* mined behaviors, datasets, and GNN architectures, Table 2 reduces each curve to its area under the curve (AUC) and reports the gap AUC(GRACE) − AUC(random). Since the area is normalized by the width of the size range and scores are clipped to $[0, 1]$, the gap is the average vertical separation between the two curves across circuit sizes. A gap of 0 means GRACE is no better than a random circuit of the same size, and the gap cannot exceed $1-$AUC(random). The gap is positive in all 24 dataset–architecture cells for both metrics, averaging 0.45 for sufficiency and 0.49 for necessity: over the range of sizes considered, GRACE's curve lies roughly half the full scale above random. Together with the saturation in Fig. 2, this indicates that the behaviors GRACE identifies are carried by compact circuits, a small fraction of the model's edges, rather than being diffused across the network. The trend holds for all four GNN architectures, whose aggregation schemes range from sum and mean to attention and concatenation. This demonstrates GRACE is robust across message-passing designs.

4.3 RQ2: DOMAIN VALIDITY

Ground truth for behaviors is generally unavailable in graph datasets, which is one of the obstacles GRACE is built to work around. MUTAGENICITY is an exception: Kazius et al. (2005) identify eight toxicophores, substructures associated with MUTAGENICITY, in the molecules of this dataset (reproduced in App. J). This lets us check GRACE's behaviors against domain knowledge. GRACE identifies six behaviors in the mutagen class. The signatures of the first five exactly match known toxicophores (Fig. 3), but their recovery is architecture-specific: GIN identifies aromatic nitro groups, GraphSAGE identifies organic azides, and GCN identifies nitrosamines, aromatic azo compounds, and acid chlorides. The sixth is recovered independently by every architecture and differs in kind. Its signature contains no single ring structure but a set of short carbon-backbone fragments that jointly cover fused, conjugated ring systems. Because these fragments recur across many such systems, the behavior detects polycyclic aromatic systems in general, which Kazius et al. (2005) also list as a toxicophore. It also shows why a signature is a set of patterns: no single fragment characterizes the behavior, but together they do. Models trained on the same data can thus rely on different subsets of the known toxicophores, and GRACE makes visible which ones each model uses.

[2]The curves in Fig. 2 do not rise monotonically with circuit size. Sufficiency dips before recovering (NCI1, near 10^{-1}), and necessity overshoots 1 (clipped at 1 in the figure) on OGBG-MOLHIV, PROTEINS. Two effects account for this, inhibitory components and destructive interference, which we discuss in Appendix N.

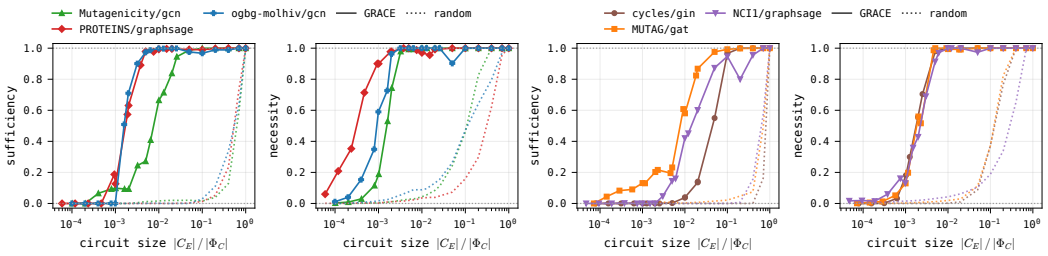
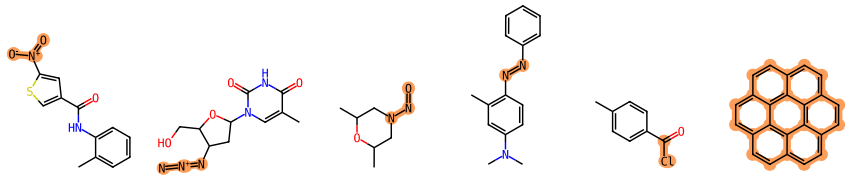


Figure 2: Sufficiency and necessity against circuit size, one behavior per dataset in GCN. The x-axis is circuit size as a fraction of $|\Phi_C|$ on a log scale.



aromatic nitro azide nitrosamine aromatic azo acid chloride conjugated ring

Figure 3: Behaviors GRACE discovers on MUTAGENICITY, one representative molecule each; orange marks the behavior's signature subgraph (§ 3.1), i.e., the corruption mask.

Table 3: Sharing of hidden units across the circuits of different behaviors in GIN. k is the number of behaviors, and Used is the number of units touched by at least one circuit, out of all units. Mono. and Poly. are the percentages of used units touched by exactly one behavior's circuit and by two or more; Dead is the percentage of all units touched by none. The per-map columns give Poly. for each linear map.

						Poly. (%) per linear map					
			% of used			Layer 1		Layer 2		Layer 3	
Dataset	k	Used	Mono.	Poly.	Dead (%)	W_1	W_2	W_1	W_2	W_1	W_2
PROTEINS	7	127/384	13	87	67	76	95	93	94	79	86
NCI1	9	191/384	19	81	50	78	98	89	67	25	68
MUTAGENICITY	8	191/384	17	83	50	79	97	89	73	43	88
MOLHIV	5	107/192	16	84	44	73	94	86	88	69	94

4.4 RQ3: CIRCUIT ANALYSIS – POLYSEMANTICITY VS. MONOSEMANTICITY

Table 3 reports how the hidden units used by circuits (256 edges each, 1–4% of Φ_C) are shared across behaviors: a unit (neuron) is *monosemantic* if it is part of only one behavior's circuit, *polysemantic* if it is part of several, and *dead* if it is part of none. Three observations follow. First, sharing is the norm: 81–87% of used units are polysemantic, and sharing peaks at the output of the first layer (94–98%), suggesting low-level features reused by every behavior. Second, the behaviors of a model jointly occupy a small part of it: 44–67% of units are dead. Third, this sharing shapes how GNNs should be analyzed and edited. Since most neurons a behavior relies on also serve others, assigning each neuron a single concept gives an incomplete picture of what it computes, and knocking out neurons to suppress one behavior risks side effects on the rest; behaviors remain separable only at the level of edge-level circuits.

The same pattern holds across circuit sizes (App. Fig. 7). Polysemantic units are already comparable in number to monosemantic ones at the smallest sizes, where few units are used and overlap by coincidence is unlikely. As circuits grow, the polysemantic share keeps rising while the monosemantic share stays at 1–13% of hidden units: additional edges recruit units shared with other behaviors rather than dedicated ones. Neuron-level interventions therefore become less selective the more completely a behavior is captured.

Additional Experiments: For further results on the efficiency of GRACE and an ablation study on the choice of clustering algorithm, we refer to App H and App. L respectively.

5 CONCLUSION

We presented GRACE, a framework for mechanistic circuit discovery in GNNs. Its central premise is that the obstacle to mechanistic analysis of GNNs lies not in circuit finding but in the steps that precede it: identifying the behaviors a model has learned, isolating the graphs that exhibit them, and corrupting them faithfully. GRACE addresses these steps through unsupervised behavior discovery in the model's functional embedding space, signatures with approximation guarantees, and continuous corruption that makes gradient-based attribution patching applicable to graphs. Across six datasets and four architectures, the recovered circuits are sufficient and necessary at a small fraction of the model's edges and outperform size-matched random circuits in every setting. On MUTAGENICITY, the behaviors correspond to known toxicophores, and circuit analysis reveals that behaviors share most of the hidden units they use, structure that remains invisible to input-level explanations.

Limitations. GRACE currently targets behaviors expressed by the presence of a substructure, in graphs with categorical node features, and its results depend on the granularity of clustering and the corruption strength; we discuss these limitations and directions for future work in App. A.

AI USE STATEMENT

Large language models were used in two capacities in this work: to refine the writing of the paper, and to assist with the implementation of the code. In both cases, the models acted on the authors' direction. The research questions, the methodology, the theoretical results, and the design of the experiments are the authors' own, and every AI-assisted passage and code change was reviewed and verified by the authors. The authors take full responsibility for the contents of this paper.

ETHICS STATEMENT

This work uses only publicly available benchmark datasets and involves no human subjects or personal data. GRACE aims to make the decisions of GNNs more transparent, which can help practitioners detect when a model relies on spurious or domain-inconsistent patterns; since recovered circuits are approximations of the model's computation, they should complement rather than replace domain validation in high-stakes applications such as drug discovery.

REPRODUCIBILITY STATEMENT

Our codebase is available at `https://anonymous.4open.science/r/grace-anonymous-1435/`. All our datasets are drawn from publicly available repositories and benchmarks. §G details the experimental setup, dataset and model specifics, and the training details.

REFERENCES

Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pp. 100–105, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136749. doi: 10.1145/780542.780558. URL https://doi.org/10.1145/780542.780558.

Burouj Armgaan, Manthan Dalmia, Sourav Medya, and Sayan Ranu. Graphtrail: Translating GNN predictions into human-interpretable logical rules. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fzlMza6dRZ.

Burouj Armgaan, Eshan Jain, Harsh Pandey, Mahesh Chandran, and Sayan Ranu. Gnnxemplar: Exemplars to explanations - natural language rules for global GNN interpretability. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=eafIjoZAHm.

Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Lio, and Andrea Passerini. Global explainability of GNNs via logic combination of learned concepts. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OTbRTIY4YS.

Mohit Bajaj, Lingyang Chu, Zi Yu Xue, Jian Pei, Lanjun Wang, Peter Cho-Ho Lam, and Yong Zhang. Robust counterfactual explanations on graph neural networks. In *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wGmOLwb8ClT.

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7/.

Leonard Bereska and Stratis Gavves. Mechanistic interpretability for AI safety - a review. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ePUVetPKu6. Survey Certification, Expert Certification.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 (suppl_1), 2005. URL https://doi.org/10.1093/bioinformatics/bti1007.

Zhihan Cheng, Yue Wu, Yule Li, Lingfeng Cai, and Baha Ihnaini. A comprehensive review of explainable artificial intelligence (xai) in computer vision. *Sensors*, 25(13):4166, 2025.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=89ia77nZ8u.

L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004. doi: 10.1109/TPAMI.2004.75.

Enyan Dai and Suhang Wang. Towards self-explainable graph neural network. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021. URL https://doi.org/10.1145/3459637.3482306.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

Joshua Engels, Eric J Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not all language model features are one-dimensionally linear. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=d63a4AM4hb.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygDF6NFPB.

Gregoire Fournier and Sourav Medya. COMRECGC: Global graph counterfactual explainer through common recourse. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qR4HCCAIf3.

Chuqin Geng, Ziyu Zhao, Zhaoyue Wang, Haolin Ye, Yuhe Jiang, and Xujie Si. Logicxgnn: Grounded logical rules for explaining graph neural networks. In *International Conference on Learning Representations*, 2026.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=TZ0CCGDcuT.

Jesse He, Helen Jenne, Max Vargas, Davis Brown, Gal Mishne, Yusu Wang, and Henry Kvinge. Minar: Mechanistic interpretability for neural algorithmic reasoning. *arXiv preprint arXiv:2602.21442*, 2026.

Yinhan He, Wendy Zheng, Yushun Dong, Yaochen Zhu, Chen Chen, and Jundong Li. Towards global-level mechanistic interpretability: A perspective of modular circuits of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=do5vVfKEXZ.

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 *Advances in Neural Information Processing Systems*, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf.

Zexi Huang, Mert Kosan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Global counterfactual explainer for graph neural networks. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, 2023. URL https://doi.org/10.1145/3539597.3570376.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Jaykumar Kakkad, Jaspal Jannu, Kartik Sharma, Charu Aggarwal, and Sourav Medya. A survey on explainability of graph neural networks. *arXiv preprint arXiv:2306.01958*, 2023.

Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 1972.

Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Mert Kosan, Samidha Verma, Burouj Armgaan, Khushbu Pahwa, Ambuj Singh, Sourav Medya, and Sayan Ranu. GNNX-BENCH: Unravelling the utility of perturbation-based GNN explainers through in-depth benchmarking. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VJvbOSXRUq.

Michihiro Kuramochi and George Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE transactions on Knowledge and Data Engineering*, 2004.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Luca Longo, Mario Brcic, Federico Cabitza, Jaesik Choi, Roberto Confalonieri, Javier Del Ser, Riccardo Guidotti, Yoichi Hayashi, Francisco Herrera, Andreas Holzinger, et al. Explainable artificial intelligence (xai) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion*, 106:102301, 2024.

Ana Lucic, Maartje A. Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, 2022. URL https://proceedings.mlr.press/v151/lucic22a.html.

Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e37b08dd3015330dcbb5d6663667b8b8-Abstract.html.

Qing Lyu, Marianna Apidianaki, and Chris Callison-Burch. Towards faithful model explanation in nlp: A survey. *Computational Linguistics*, 50(2):657–723, 2024.

Simon C Marshall and Jan H Kirchner. Understanding polysemanticity in neural networks through coding theory. *arXiv preprint arXiv:2401.17975*, 2024.

Siqi Miao, Mia Liu, and Pan Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. URL https://proceedings.mlr.press/v162/miao22a.html.

Neel Nanda. Actually, othello-gpt has a linear emergent world representation. *Neel Nanda's Blog*, 8, 2023.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, 2023.

Masahiro Negishi, Thomas Gärtner, and Pascal Welke. WILTing trees: Interpreting the distance between MPNN embeddings. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=lflqQWP1jy.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 1978.

Siegfried Nijssen and Joost N. Kok. The gaston tool for frequent subgraph mining. In *GraBaTs*, 2005.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Alessio Ragno, Marc Plantevit, and Céline Robardet. On logic-based self-explainable graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=OtAiYPP6GA.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. A practical review of mechanistic interpretability for transformer-based language models. *CoRR*, abs/2407.02646, 2024. URL https://doi.org/10.48550/arXiv.2407.02646.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In *The 7th BlackboxNLP Workshop*, 2024. URL https://openreview.net/forum?id=RysbaxAnc6.

Wenxin Tai, Ting Zhong, Goce Trajcevski, and Fan Zhou. Redundancy undermines the trustworthiness of self-interpretable GNNs. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=hFvp9NYfY9.

Wenxin Tai, Ting Zhong, Goce Trajcevski, and Fan Zhou. Self-consistency improves the trustworthiness of self-interpretable GNNs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hxGdAUn3sB.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. In *Advances in Neural Information Processing Systems*, 2020.

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3), 2008. URL https://doi.org/10.1007/s10115-007-0103-5.

Walter D Wallis, Peter Shoubridge, M Kraetz, and D Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6-7):701–704, 2001.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Xiaoqi Wang and Han Wei Shen. GNNInterpreter: A probabilistic generative model-level explanation for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=rqq6Dh8t4d.

Yuwen Wang, Shunyu Liu, Tongya Zheng, Kaixuan Chen, and Mingli Song. Unveiling global interactive patterns across graphs: Towards interpretable graph neural networks. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, 2024. URL https://dl.acm.org/doi/10.1145/3637528.3671838.

James C. R. Whittington, Will Dorrell, Surya Ganguli, and Timothy Behrens. Disentanglement with biological constraints: A theory of functional cell types. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9Z_GfhZnGH.

Yingxin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. Discovering invariant rationales for graph neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=hGXij5rfiHw.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical Science*, 9(2), 2018. URL https://doi.org/10.1039/c7sc02664a.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 2002.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in Neural Information Processing Systems*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/d80b7040b773199015de6d3b4293c8ff-Paper.pdf.

Cibo Yu, Tingting Zhu, Tingyang Chen, Yinghui Wu, Arijit Khan, and Xiangyu Ke. Slicegx: Layer-wise gnn explanation with model-slicing. In *Proceedings of the ACM Web Conference 2026*, 2026. URL https://doi.org/10.1145/3774904.3792689.

Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=bM4Iqfg8M2k.

Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. URL https://doi.org/10.1145/3394486.3403085.

Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*, pp. 12241–12252. PMLR, 2021.

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(5): 5782–5799, 2022.

He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods, and trends. *Proceedings of the IEEE*, 112(2):97–139, 2024.

Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. Protgnn: Towards self-explaining graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, 2022.

APPENDIX CONTENTS

A Limitations & Future Work **16**

B Related Work **16**

C Behavior Characterization **18**

C.1 Monotonicity and submodularity 18

C.2 Hardness 18

C.3 Greedy selection 19

C.4 Restricting the candidate pool 20

D Attribution Patching **20**

E NP-hardness of Circuit Discovery **20**

F Circuit Extraction **22**

G Experimental Setup **23**

G.1 Datasets 23

G.2 Models 24

H Running Time **24**

I Behavior consistency measures **25**

J Kazius's toxicophores **26**

K Polysemantic hidden units **26**

L Choice of Clustering Algorithm **26**

M Additional Necessity and Sufficiency Curves **30**

N Non-monotone necessity and sufficiency curves **30**

A LIMITATIONS & FUTURE WORK

The primary limitation of our framework concerns the class of graphs it currently supports. The discriminative subgraph mining step requires categorical node features, as subgraph isomorphism, which underpins the matching of discriminative patterns across graphs, is well-defined only when node identity can be established through discrete labels. Graphs with continuous or richly attributed node features, where individual features carry distinct semantic meaning, fall outside the scope of the current pipeline.

This restriction has a downstream consequence for corruption. For categorically featured graphs, the discriminative subgraph provides a clean handle: the nodes and edges to be dampened are unambiguously identified by the mining step, and this identification generalizes across domains because subgraph isomorphism is domain-agnostic. For richly featured graphs, no such universal handle exists. Consider a citation network, where a GNN may rely on homophily, the tendency of connected nodes to share topics, as its primary behavioral signal. Corrupting this behavior requires disrupting the feature similarity between neighbors, a perturbation that is semantically meaningful only within the citation domain. The same corruption applied to a road network, or a shopping network, would be entirely inappropriate. Each domain imposes its own semantic constraints on what constitutes a faithful corruption, and automating this reasoning is non-trivial. Such corruptions may therefore need to be designed manually on a domain-by-domain basis.

Armgaan et al. (2025) shows that class-level signatures can be recovered on such datasets: it selects exemplar nodes in the model's representation space and expresses what they share as Python predicate functions, which are then verbalized as natural-language rules. Such a rule delineates a set of graphs the GNN treats alike, which is what our behaviors are. So the behaviors can be identified, however not as subgraphs. Corruption remains tricky. How do you design a unique corruption for a pythonic rule? Multiple aspects can break a binary python function. Who do you corrupt? It states which property the graphs share, not which nodes, edges or feature dimensions to dampen in order to remove that property while leaving the rest of the graph intact, and for continuous features that operator is domain-specific in the way described above.

A second limitation is that GRACE must commit to a clustering algorithm, and how much that commitment matters is decided by the embedding space rather than by the algorithm. Clustering algorithms differ in what they take a cluster to be: k-means and Ward favor compact, convex groups, spectral clustering follows connectivity, and HDBSCAN looks for dense regions separated by sparse ones. Where the model's embeddings hold well-separated groups these notions coincide, the algorithms agree, and the behaviors GRACE reports are a property of the model rather than of the clustering; where they do not, the same embeddings admit several defensible partitions and the choice shapes which behaviors, and hence which circuits, are reported. Appendix L measures both regimes by swapping HDBSCAN for k-means, agglomerative or spectral clustering. On cycles the four sets of curves lie on top of one another and every algorithm mines the identical pattern; on Mutagenicity they group the graphs differently yet still reach the same chemistry. The algorithms disagree on PROTEINS: their mined patterns are different. We recommend using clustering diagnostics: coverage, the lift of the mined patterns and internal validity measures such as the silhouette score are all available before any circuit is extracted, so candidate algorithms can be compared on them and the best one for a given embedding space adopted.

Finally, our framework inherits the standard assumption of mechanistic circuit discovery that behaviors are discrete and separable. In practice, a GNN may implement highly entangled computational strategies that do not decompose cleanly into independent behaviors and circuits. In such cases, the circuits recovered by our pipeline should be interpreted as approximations rather than exact mechanistic accounts.

B RELATED WORK

We organize the discussion along the standard division into local explainers, global explainers and self-explainable architectures, and then position GRACE against the mechanistic interpretability literature it adapts.

Local explainers. The dominant formulation asks, for a single input graph, which substructure of that graph was decisive for its prediction. The methods are too numerous to enumerate individually; we only name the representatives that mark out the design space. We point reader to these surveys (Yuan et al., 2022), (Kakkad et al., 2023) which catalogue them, and GNNX-BENCH (Kosan et al., 2024) benchmarks the perturbation-based ones against one another. *Factual* explainers return the substructure that best supports the prediction, differing chiefly in how they search for it: by learning a soft mask over edges and features, as GNNExplainer (Ying et al., 2019) does and PGExplainer (Luo et al., 2020) amortizes into a generative model, or by searching subgraphs directly under a game-theoretic or probabilistic score (Yuan et al., 2021; Vu & Thai, 2020). *Counterfactual* explainers ask the complementary question of which minimal edit flips the prediction (Lucic et al., 2022; Bajaj et al., 2021). Across both branches the explanation is a subgraph of the input, and the model enters only as an oracle to be queried under perturbation.

Global explainers. A second line seeks explanations that hold across a class rather than for one graph. XGNN (Yuan et al., 2020) and GNNInterpreter (Wang & Shen, 2023) generate a synthetic graph that maximally activates a target class, by reinforcement learning over graph edits and by a probabilistic generative model respectively. Others summarize real graphs instead of synthesizing new ones: GLGExplainer (Azzolin et al., 2023) combines learned concepts into logic formulae, GraphTrail (Armgaan et al., 2024) distills the GNN into logical rules over subgraph counts via Shapley-scored decision trees, GIP (Wang et al., 2024) clusters graphs and reads off interactive patterns shared within a cluster, and GnnXemplar (Armgaan et al., 2025) selects exemplar nodes in representation space and verbalizes what they share as natural-language rules. Counterfactual reasoning has a global counterpart too, in GCFExplainer (Huang et al., 2023) and COMRECGC (Fournier & Medya, 2025), which search for a small set of recourse rules covering many inputs at once. These methods do use the representation space, but their output remains a statement about inputs: which structures a class is characterized by, not which parameters compute the characterization.

Self-explainable GNNs. A third family builds the explanation into the architecture, so that a prediction and its rationale are produced together and no post-hoc search is needed. SEGNN (Dai & Wang, 2021) predicts a node's label from its nearest labeled neighbors, making those neighbors the explanation; ProtGNN (Zhang et al., 2022) classifies by similarity to learned prototype subgraphs; and the information-bottleneck line — GIB (Yu et al., 2021), GSAT (Miao et al., 2022) and DIR (Wu et al., 2022) — trains a stochastic selector that keeps a minimal, label-sufficient (and, in DIR's case, distribution-invariant) subgraph and classifies from that alone. A logic-based strand makes the rationale symbolic rather than structural: LogiX-GIN (Ragno et al., 2025) inserts logic layers into the architecture so that the learned rules *are* the classifier, while LogicXGNN (Geng et al., 2026) extracts rules post-hoc over predicates built to mirror the GNN's message-passing structure, and argues that rule-based explanations must be scored in the final graph form rather than in an intermediate concept space. Because the bottleneck is architectural, self-explainable rationales are faithful by construction rather than by post-hoc verification, which is their principal advantage over the two families above — though recent work qualifies this, tracing the inconsistency of these built-in explanations across training seeds to redundancy under weak conciseness constraints (Tai et al., 2025) and recovering faithfulness by optimizing explanation self-consistency directly (Tai et al., 2026). The rationale is nonetheless still a subgraph of the input.

This is the common ceiling of all three families: they operate at the *input* level. Whether the explanation is a mask, a generated prototype, a logical rule or an architecturally selected rationale, it answers *which part of the graph* matters and leaves the computation that consumes it untouched. The internals of Φ — its channels, its weights, and the paths along which information travels from the input features to the logits — are never the object of study. The closest departure we are aware of is SliceGX (Yu et al., 2026), which cuts the model into layer blocks and explains each block's intermediate representation in turn; this localizes an explanation in *depth*, but each slice is still explained by a subgraph of the input, not by the weights that connect one layer's channels to the next. GRACE addresses exactly that gap: it takes the behaviors such input-level methods identify as a starting point and asks which subgraph of the *model* implements them.

Mechanistic interpretability. The mechanistic program surveyed in §1 and by (Bereska & Gavves, 2024) and (Rai et al., 2024) takes internals as the object of study, and has largely developed on transformers, where circuits have been identified for concrete behaviors such as indirect object

identification (Wang et al., 2023) and where the search itself has been progressively automated, from activation patching under a causal graph (Conmy et al., 2023) to gradient-based attribution patching and its integrated-gradients correction (Syed et al., 2024; Hanna et al., 2024) and modular variants (He et al., 2025). Graph learning has so far been approached with the earlier, coarser tools of this tradition: probing classifiers (Belinkov, 2022) applied to embeddings, and analyses of what the embedding space encodes, such as the tree metrics Negishi et al. (Negishi et al., 2025) fit to MPNN distances. The one circuit-level study we are aware of, MINAR (He et al., 2026), examines neural algorithmic reasoners, where the target behavior is given by the algorithm the network was trained to execute, so the identification, isolation and corruption steps that this paper is about do not arise. To our knowledge, GRACE is the first circuit-discovery framework for GNNs trained on real graph classification tasks.

C BEHAVIOR CHARACTERIZATION

Throughout, fix a cluster and write $\mathcal{S} = \mathcal{S}_i$, $\overline{\mathcal{S}} = \mathcal{G} \setminus \mathcal{S}_i$, $n = |\mathcal{S}|$, $\bar{n} = |\overline{\mathcal{S}}|$, and $\mathcal{P} = \mathcal{S} \times \overline{\mathcal{S}}$. For a pattern set $\mathcal{F}$, let $U(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} R(f)$ denote the pairs it resolves, so that $q(\mathcal{F}) = |U(\mathcal{F})|/|\mathcal{P}|$.

C.1 MONOTONICITY AND SUBMODULARITY

Lemma 1 *q is normalized ($q(\emptyset) = 0$), monotone, and submodular on $2^{\mathcal{D}}$.*

PROOF. q is a coverage function scaled by the constant $1/|\mathcal{P}|$: each pattern f covers the set $R(f) \subseteq \mathcal{P}$, and $q(\mathcal{F})$ is the normalized size of the union. Normalization holds since $U(\emptyset) = \emptyset$. Monotonicity holds since $\mathcal{F} \subseteq \mathcal{F}'$ implies $U(\mathcal{F}) \subseteq U(\mathcal{F}')$. For submodularity, let $\mathcal{F} \subseteq \mathcal{F}' \subseteq \mathcal{D}$ and $f \in \mathcal{D} \setminus \mathcal{F}'$. The marginal gains are

$$q(\mathcal{F} \cup \{d\}) - q(\mathcal{F}) = \frac{|R(f) \setminus U(\mathcal{F})|}{|\mathcal{P}|} \;\geq\; \frac{|R(f) \setminus U(\mathcal{F}')|}{|\mathcal{P}|} = q(\mathcal{F}' \cup \{d\}) - q(\mathcal{F}'),$$

where the inequality follows from $U(\mathcal{F}) \subseteq U(\mathcal{F}')$.

C.2 HARDNESS

Proposition 3 *Problem 2 is NP-hard. Moreover, for every $\epsilon > 0$, no polynomial-time algorithm achieves an approximation ratio of $1 - 1/e + \epsilon$ unless $\mathrm{P} = \mathrm{NP}$. Both statements hold even when every candidate pattern is a single node.*

PROOF. We reduce from MAXIMUM COVERAGE: given a universe $U = \{u_1, \ldots, u_n\}$, a family $\mathcal{A} = \{A_1, \ldots, A_m\}$ of subsets of U, and a budget s, find $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \leq s$ maximizing $|\bigcup_{A \in \mathcal{A}'} A|$. Its decision version is NP-complete, as it contains Set Cover (Karp, 1972) as the special case where the target equals n, and Alon et al. (2003) shows that it admits no polynomial-time $(1 - 1/e + \epsilon)$-approximation unless $\mathrm{P} = \mathrm{NP}$.

Construction. Use $m + 1$ node labels $\ell_\star, \ell_1, \ldots, \ell_m$.

1. **Candidate pool.** $\mathcal{D} = \{f_1, \ldots, f_m\}$, where f_i is a single node with label ℓ_i.
2. **Cluster.** For each item u_j, the graph $G_j \in \mathcal{S}$ is a path whose nodes carry the label $\ell_\star$ and the labels $\{\ell_i : u_j \in A_i\}$, one node per label.
3. **Complement.** $\overline{\mathcal{S}} = \{G_0\}$, where G_0 is a single node with label $\ell_\star$.

The instance has $n + 1$ graphs with at most $m + 1$ nodes each and m patterns, so it is built in polynomial time.

Correspondence. A single-node pattern f_i is contained in a graph iff the graph has a node labeled ℓ_i. Hence $f_i \sqsubseteq G_j$ iff $u_j \in A_i$, and $f_i \not\sqsubseteq G_0$ for every i. Therefore $R(f_i) = \{(G_j, G_0) : u_j \in A_i\}$, and since $|\mathcal{P}| = n$, for every index set $I \subseteq \{1, \ldots, m\}$

$$q\big(\{f_i : i \in I\}\big) = \frac{1}{n} \Big| \bigcup_{i \in I} A_i \Big|.$$

Algorithm 1 Greedy behavior characterization

Require: cluster $\mathcal{S}$, complement $\overline{\mathcal{S}}$, candidate pool $\mathcal{D}$, budget s
Ensure: pattern set $\mathcal{F}^\dagger$ with $|\mathcal{F}^\dagger| \le s$

for all $f \in \mathcal{D}$ **do**
  $O(f) \leftarrow \{G_a \in \mathcal{S} : f \sqsubseteq G_a\}$; $\bar{O}(f) \leftarrow \{G_b \in \overline{\mathcal{S}} : f \sqsubseteq G_b\}$ ▷ occurrence sets
end for
for all $G_a \in \mathcal{S}$ **do**
  $W_a \leftarrow \overline{\mathcal{S}}$ ▷ complement graphs not yet separated from G_a
end for
$\mathcal{F}^\dagger \leftarrow \emptyset$
for $t = 1, \dots, s$ **do**
  for all $f \in \mathcal{D} \setminus \mathcal{F}^\dagger$ **do**
    $\Delta(f) \leftarrow \sum_{G_a \in O(f)} |W_a \setminus \bar{O}(f)|$ ▷ newly resolved pairs
  end for
  $f^* \leftarrow \arg\max_{f \in \mathcal{D} \setminus \mathcal{F}^\dagger} \Delta(f)$ ▷ ties: larger $|O(f)|$
  if $\Delta(f^*) = 0$ **then break**
  end if
  $\mathcal{F}^\dagger \leftarrow \mathcal{F}^\dagger \cup \{f^*\}$
  for all $G_a \in O(f^*)$ **do**
    $W_a \leftarrow W_a \cap \bar{O}(f^*)$
  end for
end for
return $\mathcal{F}^\dagger$

The map $A_i \mapsto f_i$ is a size-preserving bijection between subfamilies of $\mathcal{A}$ and subsets of $\mathcal{D}$, under which objective values differ by the constant factor $1/n$. An exact algorithm for Problem 2 therefore solves MAXIMUM COVERAGE, and an α-approximation for Problem 2 yields an α-approximation for MAXIMUM COVERAGE. Both claims follow.

C.3 GREEDY SELECTION

Proposition 4 ((Nemhauser et al., 1978)) *For a normalized, monotone, submodular set function, the greedy algorithm that starts from $\emptyset$ and repeatedly adds the element of largest marginal gain returns, after s steps, a set $\mathcal{F}^\dagger$ with $q(\mathcal{F}^\dagger) \ge (1 - 1/e) \max_{|\mathcal{F}| \le s} q(\mathcal{F})$.*

By Lemma 1, Theorem 4 applies to q, and by Theorem 3 its ratio cannot be improved in polynomial time. Together these establish Proposition 1. The bound is relative to the optimum over $\mathcal{D}$; Proposition C.4 relates it to the optimum over all connected patterns.

Description. Algorithm 1 implements greedy selection on the pair set $\mathcal{P}$ without materializing it. For each cluster graph G_a, it maintains the set W_a of complement graphs that G_a is not yet separated from, so that $\{(G_a, G_b) : G_b \in W_a\} = \{G_a\} \times \overline{\mathcal{S}} \setminus U(\mathcal{F}^\dagger)$ at every iteration. This invariant holds initially, and adding f^* resolves exactly the pairs (G_a, G_b) with $G_a \in O(f^*)$ and $G_b \notin \bar{O}(f^*)$, which the update in the inner loop removes. Consequently, $\Delta(f)/|\mathcal{P}|$ is the marginal gain $q(\mathcal{F}^\dagger \cup \{d\}) - q(\mathcal{F}^\dagger)$, and the algorithm is the greedy algorithm of Theorem 4. A zero maximal gain means that no remaining pattern resolves a new pair, so further additions cannot increase q and the loop stops early. Since the choice at step t does not depend on s, the first t selected patterns form the greedy solution for budget t, and a single run yields the signatures for all budgets up to s.

Complexity. The occurrence sets require $|\mathcal{D}|\,|\mathcal{G}|$ subgraph isomorphism tests, computed once; $O(f)$ is typically returned by the miner itself. Representing each W_a and $\bar{O}(f)$ as a bitset of length $\bar{n}$, each gain evaluation costs $O(|O(f)|\,\bar{n}/w)$ for machine word size w, so each iteration costs $O(|\mathcal{D}|\,n\,\bar{n}/w)$ and the full run costs $O(s\,|\mathcal{D}|\,n\,\bar{n}/w)$, in addition to the isomorphism tests.

C.4 Restricting the candidate pool

Let $\mathcal{D}_{\text{all}}$ be the set of all connected patterns (up to the maximum pattern size of the miner, if one is imposed), and let $\mathcal{D} \subseteq \mathcal{D}_{\text{all}}$ be any pool such that every $f \in \mathcal{D}_{\text{all}} \setminus \mathcal{D}$ satisfies $p(f) \leq \sigma$, where $p(f)$ is the fraction of graphs in $\mathcal{S}$ that contain f. Writing $\mathrm{OPT}(\mathcal{D}) = \max_{\mathcal{F} \subseteq \mathcal{D}, |\mathcal{F}| \leq s} q(\mathcal{F})$,

$$\mathrm{OPT}(\mathcal{D}) \geq \mathrm{OPT}(\mathcal{D}_{\text{all}}) - s\sigma,$$

and Algorithm 1 run on $\mathcal{D}$ returns $\mathcal{F}^{\dagger}$ with $q(\mathcal{F}^{\dagger}) \geq (1 - 1/e)\big(\mathrm{OPT}(\mathcal{D}_{\text{all}}) - s\sigma\big)$. For any pattern f, every pair in $R(f)$ has its first graph in $O(f)$, so $|R(f)| \leq |O(f)|\,\bar{n} = p(f)\,n\,\bar{n}$ and hence $q(\{d\}) \leq p(f)$. A normalized, monotone, submodular function is subadditive, so for the optimal set $\mathcal{F}^{*}_{\text{all}}$ over $\mathcal{D}_{\text{all}}$,

$$\mathrm{OPT}(\mathcal{D}_{\text{all}}) = q(\mathcal{F}^{*}_{\text{all}}) \leq q(\mathcal{F}^{*}_{\text{all}} \cap \mathcal{D}) + \sum_{f \in \mathcal{F}^{*}_{\text{all}} \setminus \mathcal{D}} q(\{f\}) \leq \mathrm{OPT}(\mathcal{D}) + s\sigma,$$

using $|\mathcal{F}^{*}_{\text{all}} \setminus \mathcal{D}| \leq s$ and $q(\{f\}) \leq p(f) \leq \sigma$ for $f \notin \mathcal{D}$. The second claim follows by combining this with Theorem 4. The condition on $\mathcal{D}$ covers both common mining modes. With a relative support threshold σ, it holds by construction. With the k most frequent patterns, it holds for σ equal to the largest support in $\mathcal{S}$ among the excluded patterns.

D Attribution Patching

Attribution patching treats a neural network as a causal graph, in which neurons are connected by weighted edges. Within this framework, researchers perform precise interventions to isolate the components that are causally responsible for the behavior being studied. The standard protocol proceeds as follows:

1. Define a behavior exhibited by the model.
2. Devise a corruption of that behavior that leaves all other model behaviors intact.
3. Construct a dataset of instances exhibiting the behavior. For each instance, apply the corruption from the previous step to produce a negative counterpart.
4. Fix the granularity of analysis, i.e., determine what constitutes a neuron: individual rows of a layer, the layer as a whole, or the entire MLP block.
5. Treating the network's neurons and their weighted connections as a causal graph, intervene on the activations by transplanting activations from the corrupted graph into the clean graph:
   (a) Run two forward passes: one on a clean input x_{clean} and one on a corrupted input x_{corrupt}.
   (b) Replace activations at a given site in the clean run with the corresponding activations from the corrupted run.
   (c) Measure the degree to which this patching degrades the clean output. High degradation indicates that the patched component is causally important for the behavior in question.

E NP-hardness of Circuit Discovery

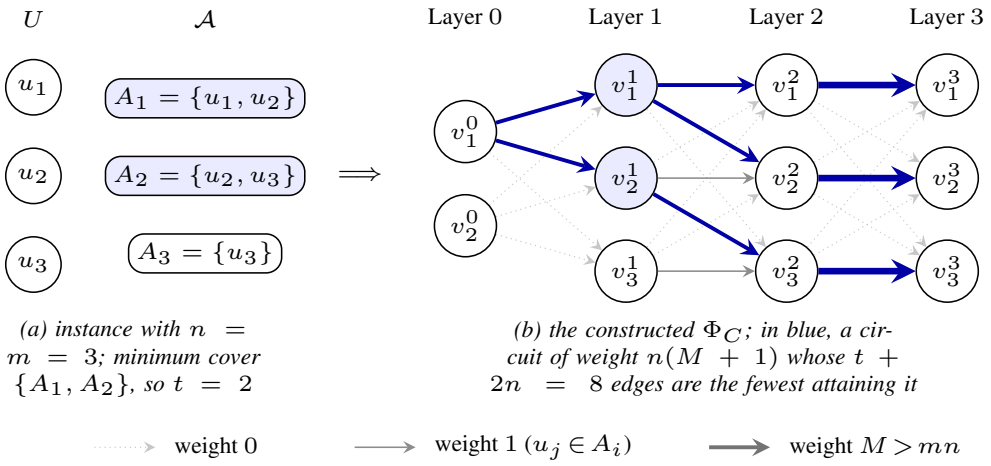
Given a budget k, our goal is to find the circuit of k edges that has the highest aggregate attribution score. We prove that finding this heaviest circuit of k edges is NP-hard.

Proof. We map the Set-Cover problem, which is NP-hard (Karp, 1972), to ours.

Definition 4 (Set Cover Problem) *Given a universe of items $U = \{u_1, \ldots, u_n\}$ and a family $\mathcal{A} = \{A_1, \ldots, A_m\}$ of subsets with $A_i \subseteq U$ and $\bigcup_{A_i \in \mathcal{A}} A_i = U$, the goal is to find the smallest subfamily of $\mathcal{A}$ that covers U, i.e.,*

$$\mathcal{A}^{*} \;=\; \arg \min_{\mathcal{A}' \subseteq \mathcal{A},\, \bigcup_{A_i \in \mathcal{A}'} A_i = U} |\mathcal{A}'|. \tag{11}$$

Given an instance $\langle U, \mathcal{A} \rangle$ of Set Cover, we construct a computation graph $\Phi_C(V, E, W)$ corresponding to it. Its neurons are arranged in four layers.



(a) instance with $n = m = 3$; minimum cover $\{A_1, A_2\}$, so $t = 2$

(b) the constructed Φ_C; in blue, a circuit of weight $n(M+1)$ whose $t + 2n = 8$ edges are the fewest attaining it

Figure 4: Reducing Set Cover to heaviest-circuit discovery. (a) A Set Cover instance with $n = m = 3$, whose minimum cover is $\{A_1, A_2\}$. (b) The constructed computation graph Φ_C, with the three weight levels distinguished by line style, and in blue a circuit of weight $n(M+1)$ whose $t+2n = 8$ edges are the fewest that attain it. Consecutive layers are fully connected, so the circuit retains only a small fraction of the edges incident on its own neurons.

1. **Layer 0** is the input layer, whose width in a GNN is the input feature dimension. We let it contain d neurons, $V_0 = \{v_1^0, \cdots, v_d^0\}$, where d is arbitrary.
2. **Layer 1** contains one neuron per subset $A_i \in \mathcal{A}$, written $V_1 = \{v_1^1, \cdots, v_m^1\}$.
3. **Layer 2** contains one neuron per item $u_j \in U$, written $V_2 = \{v_1^2, \cdots, v_n^2\}$.
4. **Layer 3** is the output layer and again contains one neuron per item, written $V_3 = \{v_1^3, \cdots, v_n^3\}$.

The node set is therefore $V = V_0 \cup V_1 \cup V_2 \cup V_3$. Consecutive layers are fully connected, as is typical of a neural network, so that $(u, v) \in E$ for every $u \in V_\ell$, $v \in V_{\ell+1}$ and $\ell \in \{0, 1, 2\}$. The instance is encoded entirely in the weights.

1. **Layer 0 to Layer 1 edges.** Every edge has weight 0.
2. **Layer 1 to Layer 2 edges.** The edge (v_i^1, v_j^2) has weight 1 if $u_j \in A_i$, and weight 0 otherwise. We refer to an edge of the former kind as a *covering* edge.
3. **Layer 2 to Layer 3 edges.** The edge (v_j^2, v_j^3) has weight $M > mn$, and every other edge between these layers has weight 0. We refer to the n edges of the form (v_j^2, v_j^3) as *matched* edges.

The construction takes polynomial time, since $|V_1| = |\mathcal{A}|$, $|V_2| = |V_3| = |U|$, $|V_0| = d$, and the number of edges is quadratic in the number of neurons because consecutive layers are fully connected. Taking $M = mn + 1$ additionally keeps every weight polynomially bounded in the size of the instance. Throughout, we write $t = |\mathcal{A}^*|$ for the size of a minimum cover. Figure 4 illustrates the construction on a three-item instance.

The two positive weight levels play distinct roles. A matched edge is the only weighted outgoing edge of its Layer 2 neuron, so it pays for an item exactly once, however many selected subsets contain that item; this is what makes the objective count covered items rather than subset-item incidences. A covering edge, in turn, is what makes a parent that contains the item preferable to one that does not.

We now show that the smallest circuit of aggregate weight at least $n(M+1)$ has size exactly $k = t + 2n$. Given this, an oracle for the heaviest circuit recovers t: the optimum weight under budget k is non-decreasing in k, so sweeping or binary-searching k over $\{0, \ldots, |E|\}$ locates the smallest budget at which the optimum first reaches $n(M+1)$, and subtracting $2n$ from that budget k yields the minimum cover size. A polynomial-time algorithm for the heaviest circuit would therefore solve Set Cover in polynomial time.

Set-Cover ⇒ circuit. We exhibit a circuit of weight $n(M+1)$ and size $t+2n$; Figure 4(b) shows the one produced for the instance of Figure 4(a). Let $V^1_{C^*} = \{v^1_i \in V_1 \mid A_i \in \mathcal{A}^*\}$ be the Layer 1 neurons of the minimum cover, so that $|V^1_{C^*}| = t$, and since $\mathcal{A}^*$ covers U we may take every Layer 2 and Layer 3 neuron into the circuit.

The edges between Layers 1 and 2 are chosen in a single pass: for each $v^1_i \in V^1_{C^*}$ and each $u_j \in A_i$, we add the edge (v^1_i, v^2_j) if v^2_j does not already have an incoming edge, and skip it otherwise. Every Layer 2 neuron thereby acquires exactly one incoming edge, which is a covering edge, and every selected Layer 1 neuron acquires at least one outgoing edge. The latter holds because a minimum cover is irredundant: each $A_i \in \mathcal{A}^*$ contains an item that no other member of $\mathcal{A}^*$ contains, and that item can only be assigned to A_i. Across Layers 2 and 3 we select the n matched edges, one per Layer 2 neuron. These $2n$ edges already carry the full weight $n \cdot 1 + n \cdot M = n(M+1)$.

It remains to give each selected Layer 1 neuron the incoming edge it requires, which we do by adding (v^0_1, v^1_i) for every $v^1_i \in V^1_{C^*}$. Routing all of them through the same input neuron also makes the circuit weakly connected, and all four layers are occupied. This adds t edges, completing a circuit of weight $n(M+1)$ and size $t+2n$.

Circuit ⇒ set-cover. Conversely, we show that every circuit C with $W(C) \geq n(M+1)$ satisfies $|E_C| \geq t+2n$. Observe first that no Layer 2 neuron can be missing from C. Matched edges are the only edges of weight M and each Layer 2 neuron carries exactly one of them, whereas all edges between Layers 1 and 2 together carry weight at most mn. Hence if $d' \geq 1$ Layer 2 neurons were absent, the aggregate weight would be at most $mn + M(n-d') < M + Mn - M = Mn$, short of the required $n(M+1)$. All n matched edges therefore lie in E_C and contribute Mn, so the residual weight of n must come from covering edges, of which there are consequently at least n.

Every Layer 2 neuron also needs a parent. Let u denote the number of them whose incoming edges all have weight 0; the remaining $n-u$ each have a covering parent, so the items they represent are covered by $\mathcal{A}' = \{A_i : v^1_i \in V^1_C\}$. In Figure 4 every Layer 2 neuron has a covering parent, so $u = 0$ there, but in general u may be positive. These u weight-0 parent edges are distinct from the n covering edges counted above, so E_C contains at least $n+u$ edges between Layers 1 and 2. Adding the $|V^1_C|$ edges that supply the Layer 1 neurons with parents, together with the n matched edges, gives

$$|E_C| \;\geq\; |V^1_C| + (n+u) + n. \tag{12}$$

Finally, the items left uncovered by $\mathcal{A}'$ are among those u items, so appending one member of $\mathcal{A}$ per uncovered item, which exists because $\bigcup_{A_i \in \mathcal{A}} A_i = U$, extends $\mathcal{A}'$ to a cover of size at most $|V^1_C| + u$. Minimality of $\mathcal{A}^*$ therefore gives $|V^1_C| + u \geq t$, and substituting into equation 12 yields $|E_C| \geq t+2n$.

Combining the two directions, the smallest circuit of aggregate weight at least $n(M+1)$ has size exactly $t+2n$, and by the argument above a polynomial-time algorithm for the heaviest circuit under a budget would solve Set Cover. Since all weights are bounded by $M = mn+1$, the hardness holds in the strong sense. □

F Circuit Extraction

We give here the extraction procedure summarized in § 3.3. Let $\Delta_e \mathcal{L}$ denote the attribution score of edge e, averaged over the behavior's cluster as described there, and let k be the edge budget. Write $\mathrm{par}(e)$ and $\mathrm{ch}(e)$ for the neurons an edge leaves and enters, and V_{out} for the neurons of the output layer.

Three properties of the procedure are worth making explicit.

No childless neurons. An edge is eligible only once its child is active, and the only neurons active at the outset are the output logits. Every neuron admitted to V_C therefore carries a signal that reaches the output along edges already in E_C, at every intermediate size and not merely at the end. Parentage is the direction the rule cannot police: if the budget is exhausted before the frontier reaches the input features, the circuit retains edges whose parents are never supplied, and it is then discarded rather than interpreted.

Algorithm 2 Backward greedy circuit extraction

Require: computation graph $\Phi_C = (V, E)$, scores $\{\Delta_e \mathcal{L}\}_{e \in E}$, budget k
Ensure: circuit $C = (V_C, E_C)$ with $|E_C| \leq k$
$V_C \leftarrow V_{\text{out}}, \quad E_C \leftarrow \emptyset$
$D \leftarrow \{e \in E : \text{ch}(e) \in V_C\}$ ▷ the frontier
while $|E_C| < k$ **and** $D \neq \emptyset$ **do**
  $e^* \leftarrow \arg\max_{e \in D} |\Delta_e \mathcal{L}|$
  $E_C \leftarrow E_C \cup \{e^*\}, \quad V_C \leftarrow V_C \cup \{\text{par}(e^*)\}$
  $D \leftarrow \{e \in E \setminus E_C : \text{ch}(e) \in V_C\}$
end while
return (V_C, E_C)

Nestedness. The edge chosen at step i depends only on the circuit built in the preceding $i - 1$ steps, never on k. The circuit at a given budget is thus a subset of the circuit at any larger one, and a single run yields the entire family $\{C^{(1)}, \ldots, C^{(k)}\}$ as prefixes. This is what makes the size sweeps of § 4.2 a study of one nested sequence rather than of k unrelated circuits, and we impose the same property on the random control they are read against.

Cost. Maintaining D as a priority queue, each edge is pushed once, when its child first becomes active, and popped at most once, giving $O(|E| \log |E|)$ overall and no dependence on k beyond the loop itself. When a neuron's input is formed by summing the outputs of several upstream layers, activating it makes the corresponding channel of each summand active, so that no branch of the computation is left permanently ineligible.

G Experimental Setup

Experiments ran on a server with an Intel(R) Xeon(R) Gold 6426Y (64 cores, 126 GB RAM) and three NVIDIA L40S GPUs (45 GiB each), running Ubuntu 20.04.6 LTS.

G.1 Datasets

cycles is synthetic (adapted from the Tree-Cycles construction of Ying et al. (2019) to graph classification): class 0 is a random tree and class 1 the same kind of tree with a 6-node cycle attached by a single edge, so a graph is mutagenic to this task, so to speak, exactly when it contains that planted cycle. Node features are a one-hot encoding of each node's degree, bucketed at $0, 1, 2, 3, 4+$; there are no edge features.

MUTAG (Debnath et al., 1991) contains aromatic and heteroaromatic nitro compounds, labeled by their mutagenic effect on *Salmonella typhimurium*. Node features are a one-hot atom type (7 types) and edge features a one-hot bond type (4 types).

PROTEINS (Borgwardt et al., 2005) represents proteins as graphs of secondary structure elements (helix, sheet, turn), edges connecting elements that are neighbors in the amino-acid sequence or in 3D space; the task is to classify a protein as an enzyme or not. Node features are a one-hot secondary-structure-element type (3 types); there are no edge features.

NCI1 (Wale et al., 2008) contains chemical compounds screened for activity against non-small-cell lung cancer. Node features are a one-hot atom type (37 types); there are no edge features.

Mutagenicity (Kazius et al., 2005) contains chemical compounds labeled by the Ames test for mutagenicity; unlike MUTAG, class 0 is the mutagen class here. Node features are a one-hot atom type (14 types) and edge features a one-hot bond type (3 types).

ogbg-molhiv (Hu et al., 2020) (originally from MoleculeNet (Wu et al., 2018)) contains molecules labeled by their ability to inhibit HIV replication. OGB provides nine integer node features and three integer edge features per graph; our mining stage needs a one-hot feature whose $\arg\max$ is itself a chemical label, so we keep only each node's atom type and each edge's bond type, one-hot encoded (14 and 4 categories respectively, the 13 most common atomic numbers plus a shared

other channel), and drop the remaining eight node features (chirality, degree, formal charge, #H, radical electrons, hybridization, aromaticity, ring membership) and two edge features. This makes the model strictly weaker than one reading OGB's full feature set through its `AtomEncoder`, which is why we treat the OGB leaderboard number as a ceiling rather than a target (Table 4). At its native 3.5% positive rate, the class used for behavior discovery is too small to survive our locality filter; behavior discovery and circuit extraction therefore run on a class-balanced 2,886-graph subsample (all 1,443 positives plus an equal number of negatives), while the model itself is trained and evaluated on the full, official split.

Table 1 reports size and feature statistics for all six; #nodes and #edges are per-graph averages, edges counted once per undirected pair.

Data splits. All datasets are binary classification. For the five TUDatasets we draw one seeded, class-stratified 80/10/10 train/val/test split; MUTAG uses 70/15/15 instead, since at 80/10/10 its test set is only 20 graphs and a single graph is worth 5 accuracy points. ogbg-molhiv instead uses OGB's official scaffold split (molecules grouped by Bemis–Murcko scaffold, whole scaffolds assigned to one part) for both training and the accuracy figures reported below, since that is what the leaderboard number measures and no seeded split reproduces it; the class-balanced subsample used for behavior discovery is drawn separately, under an ordinary seeded stratified split, and never touches the scaffold split's test molecules used for reporting accuracy.

G.2 Models

We train all four architectures – GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2019) and GraphSAGE (Hamilton et al., 2017) – on every dataset, with edge features consumed by an edge-aware wrapper (GINEConv-style) where a dataset provides them (MUTAG, Mutagenicity, ogbg-molhiv). Table 4 reports the selected configuration and test performance for each of the 24 (dataset, architecture) pairs, chosen by the grid search below. Published reference numbers, where a comparable one exists, come from Xu et al. (2019) (10-fold cross-validation mean) for GIN on MUTAG/PROTEINS/NCI1, from Errica et al. (2020) (also 10-fold) for GraphSAGE on PROTEINS/NCI1, and from the ogbg-molhiv leaderboard (Hu et al., 2020) for GCN and GIN; they are included as a sanity band rather than a target, since neither protocol matches ours exactly (§G.1).

Training procedure. All models are trained with Adam (weight decay 5×10^{-4}) for up to 1,000 epochs (200 for ogbg-molhiv, which has two orders of magnitude more graphs), with a `StepLR` schedule halving the learning rate every 300 epochs and early stopping at patience 200 after a 100-epoch floor (patience 40 for ogbg-molhiv). Model selection and early stopping are driven by validation macro-F1 on every dataset except ogbg-molhiv, where they use validation ROC-AUC instead: three datasets are class-imbalanced (MUTAG 125/63, PROTEINS 663/450, Mutagenicity 2,401/1,936) and ogbg-molhiv is 3.5% positive, at which accuracy and even macro-F1 mostly report where the decision threshold happened to fall. The epoch with the best validation score has its weights restored into the model before the test set is touched, and ogbg-molhiv additionally trains with a class-weighted loss: without it, the reported ROC-AUC is comparable but the model predicts the positive class for essentially no graph, which is fatal for a pipeline that discovers behaviors per predicted class.

Hyperparameter search. For each (dataset, architecture) pair we grid-search hidden width $H \in \{32, 64\}$, depth $L \in \{2, 3\}$ and learning rate $\in \{10^{-3}, 5\times10^{-3}\}$; on cycles, MUTAG and PROTEINS we additionally search dropout $\in \{0, 0.5\}$, while NCI1 and Mutagenicity fix dropout at 0.5 – with several thousand training graphs each, dropout is the axis least likely to matter and halves the grid. The configuration with the best validation score (macro-F1, or ROC-AUC for ogbg-molhiv) is selected and reported in Table 4.

H Running Time

We time each of the 6 granular stages of GRACE on GIN: (1) *embedding generation* through forward pass, (2) *clustering*), (3) *mining frequent subgraphs*, (4) behavior signature construction (Alg. 1), (5) *attribution patching* and (6) *greedy circuit extraction* (Alg. 2).

Table 4: Selected configuration and test-set performance of the GNNs used in our experiments. Hidden width H and depth L are chosen by the grid search of §G.2; test macro-F1 and accuracy are on the held-out split of §G.1, and test ROC-AUC is reported only for ogbg-molhiv, whose leaderboard metric it is.

Dataset	Model	H	L	Test F1	Test Acc.	Test AUC	Published
cycles	GCN	64	3	0.990	0.990	–	–
cycles	GAT	64	3	1.000	1.000	–	–
cycles	GIN	64	2	1.000	1.000	–	–
cycles	GraphSAGE	32	3	1.000	1.000	–	–
MUTAG	GCN	32	4	0.807	0.833	–	–
MUTAG	GAT	64	4	0.689	0.767	–	–
MUTAG	GIN	64	3	0.689	0.767	–	0.894 ± 0.056 (Xu et al., 2019)
MUTAG	GraphSAGE	64	2	0.689	0.767	–	–
PROTEINS	GCN	32	3	0.807	0.812	–	–
PROTEINS	GAT	64	2	0.820	0.830	–	–
PROTEINS	GIN	64	3	0.756	0.777	–	0.762 ± 0.028 (Xu et al., 2019)
PROTEINS	GraphSAGE	64	3	0.813	0.821	–	0.730 ± 0.045 (Errica et al., 2020)
NCI1	GCN	64	3	0.767	0.768	–	–
NCI1	GAT	64	3	0.760	0.760	–	–
NCI1	GIN	64	3	0.801	0.801	–	0.827 ± 0.017 (Xu et al., 2019)
NCI1	GraphSAGE	64	3	0.792	0.792	–	0.760 ± 0.018 (Errica et al., 2020)
Mutagenicity	GCN	64	3	0.762	0.766	–	–
Mutagenicity	GAT	64	3	0.758	0.761	–	–
Mutagenicity	GIN	64	3	0.782	0.784	–	–
Mutagenicity	GraphSAGE	64	3	0.759	0.764	–	–
ogbg-molhiv	GCN	64	3	0.585	0.921	0.727	0.761 ± 0.010 (Hu et al., 2020)
ogbg-molhiv	GAT	64	3	0.591	0.902	0.760	–
ogbg-molhiv	GIN	32	3	0.604	0.926	0.729	0.756 ± 0.014 (Hu et al., 2020)
ogbg-molhiv	GraphSAGE	32	3	0.575	0.883	0.755	–

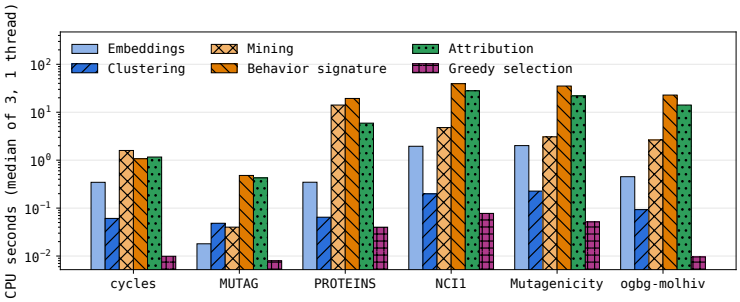


Figure 5: Median CPU seconds per pipeline stage on GIN, over three fresh-process repeats.

The whole method, from a trained model to one circuit per behavior, costs on the order of a few to a few tens of CPU-seconds on one thread, growing with the number of graphs a dataset's behaviors cover. The most expensive components are VF2 (Cordella et al., 2004) subgraph matching in behavior construction (Alg. 1), and EAP-IG attribution. Embeddings, clustering and greedy selection are comparatively negligible throughout. The trends are same across other GNN architectures since the only part that differs is the forward pass through the GNN.

I Behavior consistency measures

Tables 2 and 6 compare two behavior-discovery runs on the same analysis graphs, indexed identically, so a graph's position is a cross-run anchor even though embeddings and cluster ids are not. Each behavior i of a run is a predicted class, a set of member graphs M_i, a set of mined patterns, and

a node-level corruption mask $V_i(G)$ for every $G \in M_i$. Given two runs A and B, the four measures are computed as follows.

Matching. Behaviors are paired one to one by the Hungarian assignment maximizing $\sum J(M_i^A, M_j^B)$, where

$$J(M_i^A, M_j^B) = \frac{|M_i^A \cap M_j^B|}{|M_i^A \cup M_j^B|}$$

is graph-set Jaccard. Behaviors of different predicted classes are never paired, and a pair with $J = 0$ counts as unmatched. All four measures below are computed only over this matching, then summarized over every ordered pair of runs being compared.

Graph J. The size-weighted mean of $J(M_i^A, M_j^B)$ over A's behaviors, unmatched behaviors counting 0: whether the two runs group the same graphs together.

Pattern sim. For a matched pair, each of A's mined patterns takes its best score against B's patterns and the results are averaged, where the score between two patterns P, Q is the maximum-common-subgraph intersection-over-union

$$\mathrm{sim}(P, Q) = \frac{|\mathrm{MCS}(P, Q)|}{|P| + |Q| - |\mathrm{MCS}(P, Q)|},$$

sizes counted in edges and $\mathrm{MCS}(P, Q)$ the largest connected subgraph common to both respecting node and edge labels (Wallis et al., 2001). This is 1 exactly when the two patterns are isomorphic and 0 when they share no labelled bond, so it reads on the same $[0, 1]$ scale as the Jaccard columns; whether the two runs mine similar structure.

Sim null. The same maximum-common-subgraph score, but averaged over pairs of same-predicted-class behaviors that the Hungarian matching did *not* pair. Small carbon fragments share a bond with almost anything, so pattern sim carries evidence only where it clears this baseline.

Mask J. For a matched pair, the node-level Jaccard of $V_i^A(G)$ and $V_j^B(G)$ averaged over graphs G present in both M_i^A and M_j^B: whether the two runs end up dampening the same nodes, independent of whether their mined patterns are isomorphic.

J KAZIUS'S TOXICOPHORES

Figure 6 reproduces, for reference, the general toxicophores of Kazius et al. (2005) that Section 4 checks GRACE's Mutagenicity behaviours against.

K POLYSEMANTIC HIDDEN UNITS

For one fixed circuit size, how are a model's hidden units shared across its behaviors? A neuron is *monosemantic* if only one behavior's circuit touches it and *polysemantic* if two or more do. Table 3 reports this split for the four datasets whose GIN model has more than one behavior; MUTAG and cycles are both omitted for the same reason, each finding a single discovered behavior, which makes every used unit monosemantic by construction and the split uninformative.

L CHOICE OF CLUSTERING ALGORITHM

GRACE identifies behaviors by clustering graph embeddings (§ 3.1) but does not prescribe a clustering algorithm; our experiments use HDBSCAN. To measure how much the results depend on that choice, we rerun behavior discovery on GIN with only the clustering algorithm swapped, replacing HDBSCAN with k-means, agglomerative (Ward) or spectral clustering. Subgraph mining, the grid of circuit sizes and the random control are those of § 4.2. The three alternatives require a fixed number of clusters, which we choose per predicted class at the elbow of the within-cluster sum of

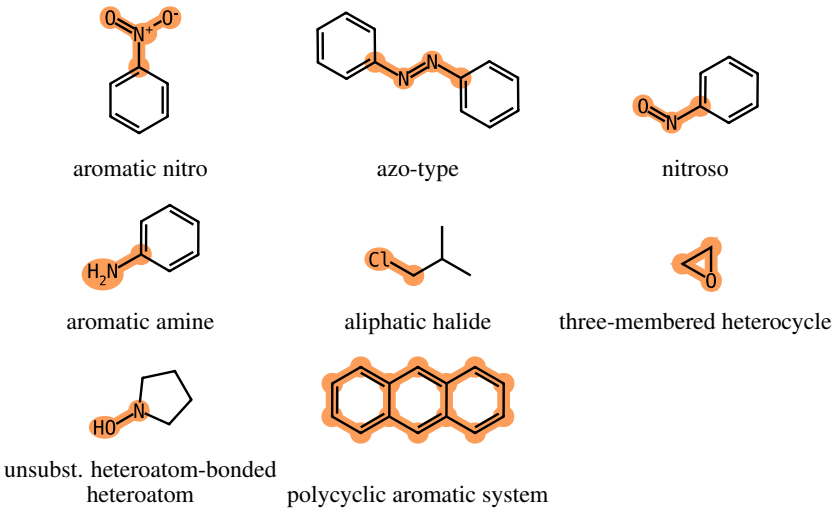


Figure 6: The eight general toxicophores of (Kazius et al., 2005); the highlighted atoms are the matched substructure. GRACE's Mutagenicity behaviours (Figure 3) match aromatic nitro, azo-type (specifically, organic azides) and, polycyclic aromatic system.

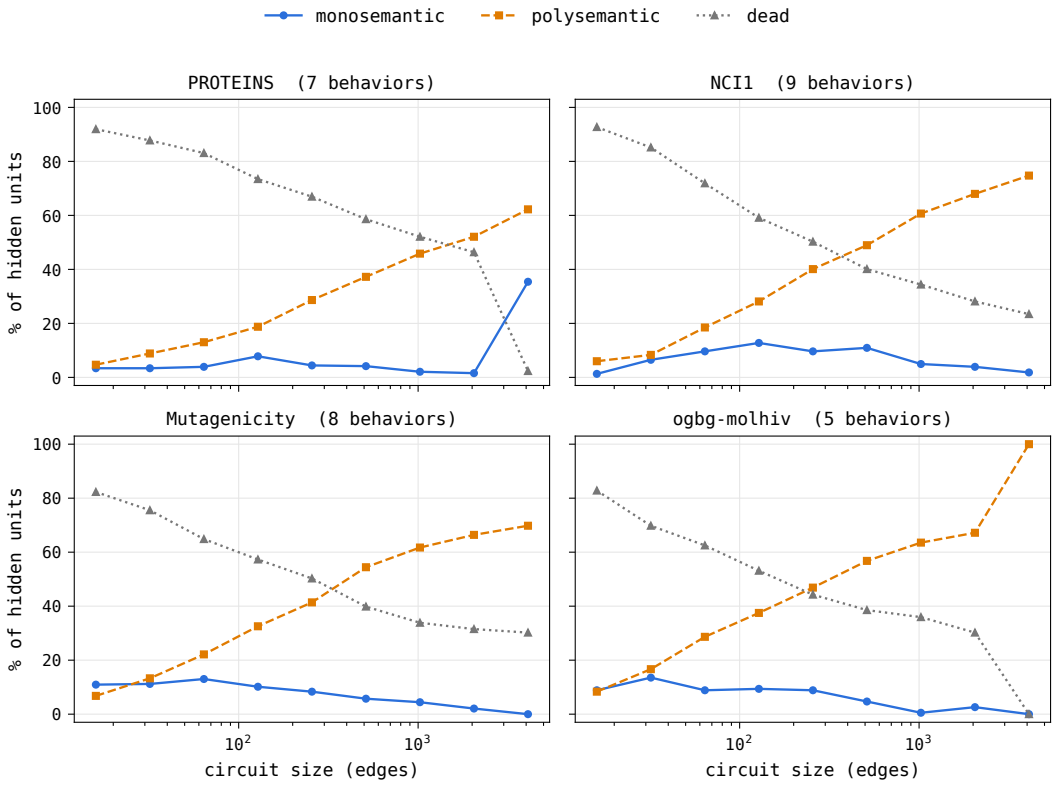


Figure 7: Monosemantic, polysemantic and dead hidden units against circuit size on GIN, each as a percentage of the model's hidden units, so the three curves of a panel sum to 100%. The number of behaviors the model has is given in parentheses.

squares over $k \in \{2, \ldots, 8\}$, without further tuning. Unlike HDBSCAN, they assign every graph to some cluster rather than setting part of the data aside as noise.

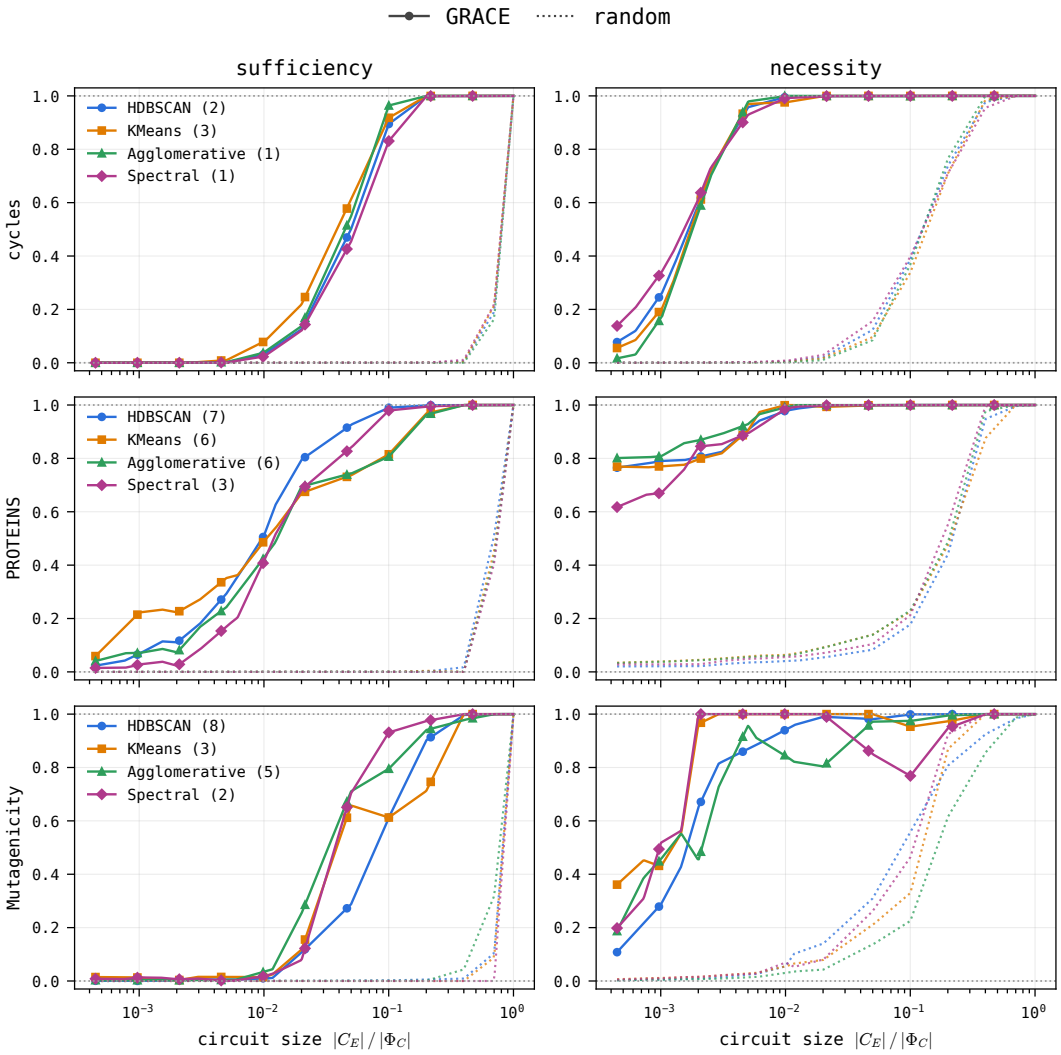


Figure 8: Sufficiency (left) and necessity (right) against circuit size on GIN under four clustering algorithms. Solid lines are GRACE's circuits and dotted lines their size-matched random controls, each averaged over the behaviors an algorithm finds, with scores clipped to $[0, 1]$. The number of behaviors is given in parentheses.

Table 5 reports, for each dataset and algorithm, the number of behaviors that survive the locality filter, their *coverage* (the fraction of graphs that belong to one of them), the mean *lift* of their mined patterns (the fraction of the behavior's graphs that contain the pattern minus the fraction of background graphs that do), and the sufficiency and necessity AUC gaps of Table 2. Figure 8 shows the underlying curves. Table 6 compares the behaviors of each pair of algorithms with the measures of §I: behaviors are matched one to one by graph-set Jaccard within each predicted class, and graph J, pattern sim and mask J report how far a matched pair agrees on its graphs, its mined subgraph and its corruption mask, with sim null as the baseline for pattern sim.

The circuits do not hinge on HDBSCAN. Under every algorithm and on every dataset, GRACE's circuits clear their size-matched random control by a wide margin: the sufficiency gaps range from 0.31 to 0.56 and the necessity gaps from 0.49 to 0.69, and none approaches zero. On PROTEINS and Mutagenicity the three fixed-k algorithms land within about 0.1 of HDBSCAN on both scores, and on Mutagenicity they slightly exceed it. The behaviors themselves largely carry over as well. On cycles the four sets of curves lie on top of one another (Figure 8), and every pair of algorithms that shares graphs mines the identical pattern (pattern sim 1.00): the dataset holds essentially one

Table 5: Behavior discovery and circuit faithfulness on GIN under four clustering algorithms. k is the number of clusters the elbow rule chose for each predicted class (HDBSCAN determines its own). The ΔAUC columns are defined as in Table 2, mean $\pm$ std over behaviors.

Dataset	Clustering	k	$\lvert\mathcal{B}\rvert$	Coverage	Lift	ΔAUC suff.	ΔAUC nec.
cycles	HDBSCAN	–	2	0.36	0.40	0.37 ± 0.02	0.55 ± 0.01
cycles	KMeans	4/4	3	0.79	0.38	0.39 ± 0.04	0.55 ± 0.01
cycles	Agglomerative	4/4	1	0.24	0.57	0.38 ± 0.00	0.54 ± 0.00
cycles	Spectral	3/4	1	0.32	0.19	0.35 ± 0.00	0.56 ± 0.00
PROTEINS	HDBSCAN	–	7	0.63	0.22	0.56 ± 0.06	0.69 ± 0.08
PROTEINS	KMeans	4/5	6	0.71	0.13	0.55 ± 0.18	0.67 ± 0.03
PROTEINS	Agglomerative	4/5	6	0.70	0.09	0.51 ± 0.13	0.68 ± 0.04
PROTEINS	Spectral	3/5	3	0.60	0.11	0.51 ± 0.03	0.65 ± 0.04
Mutagenicity	HDBSCAN	–	8	0.48	0.37	0.31 ± 0.05	0.49 ± 0.07
Mutagenicity	KMeans	4/4	3	0.72	0.10	0.34 ± 0.07	0.58 ± 0.08
Mutagenicity	Agglomerative	3/5	5	0.76	0.13	0.37 ± 0.03	0.56 ± 0.10
Mutagenicity	Spectral	4/4	2	0.41	0.06	0.40 ± 0.00	0.52 ± 0.10

Table 6: Agreement between the behaviors found by each pair of clustering algorithms on GIN, averaged over both matching directions. **Bold** marks where pattern sim exceeds its null. Dashes mark pairs that share no graphs.

Dataset	Algorithm pair	Graph J	Pattern sim	Sim null	Mask J
cycles	HDBSCAN–KMeans	0.56	**1.00**	0.17	0.95
cycles	HDBSCAN–Agglomerative	0.64	**1.00**	0.00	1.00
cycles	HDBSCAN–Spectral	0.34	**1.00**	0.00	0.90
cycles	KMeans–Agglomerative	0.65	**1.00**	0.33	1.00
cycles	KMeans–Spectral	0.59	**1.00**	0.00	0.85
cycles	Agglomerative–Spectral	0.00	–	0.00	–
PROTEINS	HDBSCAN–KMeans	0.60	**0.60**	0.45	0.58
PROTEINS	HDBSCAN–Agglomerative	0.51	0.24	0.39	0.43
PROTEINS	HDBSCAN–Spectral	0.47	0.19	0.23	0.47
PROTEINS	KMeans–Agglomerative	0.81	**0.78**	0.54	0.68
PROTEINS	KMeans–Spectral	0.31	**0.44**	0.39	0.78
PROTEINS	Agglomerative–Spectral	0.28	**0.60**	0.48	0.66
Mutagenicity	HDBSCAN–KMeans	0.47	**0.43**	0.04	0.61
Mutagenicity	HDBSCAN–Agglomerative	0.62	**0.22**	0.10	0.11
Mutagenicity	HDBSCAN–Spectral	0.36	**0.73**	0.15	0.90
Mutagenicity	KMeans–Agglomerative	0.60	**0.73**	0.26	0.54
Mutagenicity	KMeans–Spectral	0.23	**0.83**	0.42	0.81
Mutagenicity	Agglomerative–Spectral	0.33	**0.62**	0.38	0.35

discriminative motif, and every algorithm finds it. The one exception, agglomerative and spectral clustering, is not a disagreement: each retains a single behavior, and they turn out to have locked onto two different sub-clusters that HDBSCAN also keeps separate, so the two behaviors share almost no graphs and pairwise agreement has nothing to compare. Pairwise agreement need not be transitive, and HDBSCAN's own two clusters are the reason it isn't here. On Mutagenicity the algorithms group the graphs differently (graph J 0.23–0.62), yet in every pair the mined patterns agree beyond the null, for instance 0.73 against 0.15 for HDBSCAN and spectral clustering, whose corruption masks overlap by 0.90. The algorithms reach the same chemistry through different partitions.

Where the choice matters. Clustering algorithms differ in what they take a cluster to be. k-means and Ward favor compact, convex groups, spectral clustering follows connectivity, and HDBSCAN looks for dense regions separated by sparse ones. Where the embedding space holds well-separated groups, these notions coincide and the algorithms agree, as on cycles. Where it does not, they par-

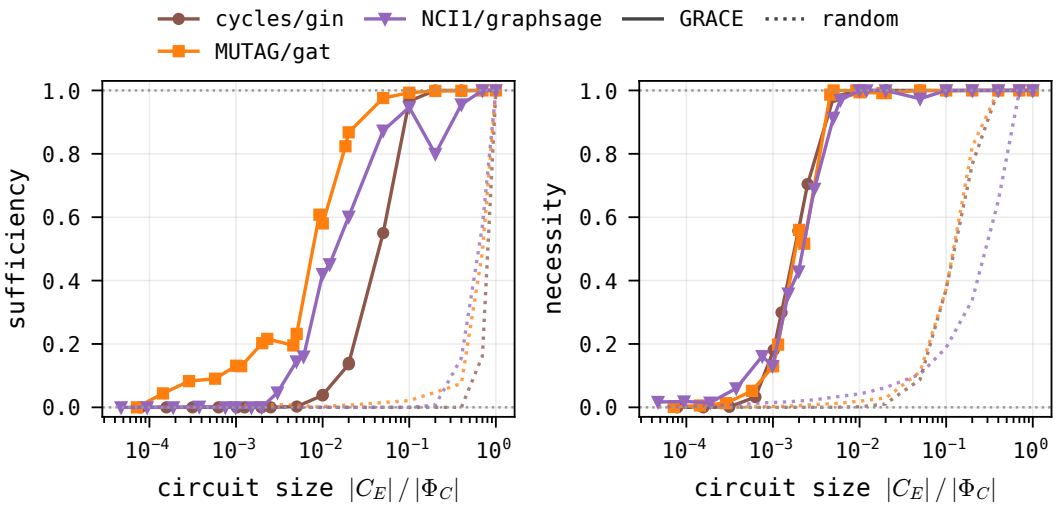


Figure 9: Sufficiency and necessity against circuit size for cycles, MUTAG and NCI1, continuing Figure 2.

tition the same embeddings differently, and GRACE then reports different behaviors. PROTEINS, with three generic node labels and weak class structure, is such a case. HDBSCAN's mined patterns agree with those of agglomerative and spectral clustering *less* than unmatched behaviors do (0.24 against 0.39, and 0.19 against 0.23; Table 6), so the algorithms find genuinely different discriminative subgraphs rather than the same one seen through different partitions. The fixed-k algorithms, which assign every graph, also return coarser behaviors on PROTEINS and Mutagenicity than HDBSCAN: they cover more of the data (0.60–0.76 against 0.48–0.63) with weaker patterns (mean lift 0.06–0.15 against 0.22–0.37; Table 5).

This is a limitation of the approach. GRACE must commit to a clustering algorithm, and on embedding spaces without clear cluster structure that commitment shapes which behaviors, and hence which circuits, are reported. The sufficiency and necessity gaps cannot reveal it, since on PROTEINS they remain high under every algorithm. The clustering's own diagnostics can, however: coverage, the lift of the mined patterns, and standard internal validity measures such as the silhouette score are all available before any circuit is extracted. A practitioner can therefore compare candidate algorithms on these measures and adopt the one that performs best on the model's embedding space.

M ADDITIONAL NECESSITY AND SUFFICIENCY CURVES

Figure 9 continues Figure 2 with the remaining three datasets, one behavior each. The setup is identical: solid lines are GRACE's circuit, dotted lines the mean of its size-matched random control shaded to ± 1 s.d., the x-axis is circuit size as a fraction of $|\Phi_C|$ on a log scale, and both scores are clipped to $[0, 1]$.

N NON-MONOTONE NECESSITY AND SUFFICIENCY CURVES

Inhibitors. Models contain components that hinder a behavior, acting as breaks, as well as components that help it. Because Eq. equation 8 ranks edges by the magnitude of their scores, inhibitory edges can enter the circuit. If a circuit at a given size includes the helpful edges but not the inhibitory ones, the circuit is running without the breaks expressing its behavior with greater confidence than the full model does. As the circuit size is grown, and the inhibitors make it into the circuit, it pulls the curve down leading to the non-monotonic behavior we observe. Such brakes are not peculiar to graphs. They have been documented in the language-model literature, most prominently as the negative name mover heads of the indirect object identification circuit (Wang et al., 2023). Hanna et al. (2024) also mentions inhibitors, and they resurface here in GNNs.

Early destructive interference. Patching transplants corrupted activations into an otherwise clean run, so an ablated model is a hybrid of the two. At very low edge counts, the hybrid can be incoherent:, if a critical pathway is broken, the hybrid mixture of clean and corrupted activations can cause the model's performance to degrade heavily. Downstream components receive a mixture of clean and corrupted signals that they never see in either run (clean and corrupt runs). In this case, the model's output distribution can become completely incoherent, resulting in a metric value that is even lower than if the model had been run on a cleanly-formatted corrupted input. In our metrics, this can show up as sufficiency falling below 0 at tiny circuit sizes, since $\sigma(C_i) < 0$ exactly when the ablated output, summed over $\mathcal{S}_i$, is further from the clean output than the fully corrupted output is.