MINAR: MECHANISTIC INTERPRETABILITY FOR NEURAL ALGORITHMIC REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

The recent field of *neural algorithmic reasoning* (NAR) studies the ability of graph neural networks (GNNs) to emulate classical algorithms like Bellman-Ford, a phenomenon known as *algorithmic alignment*. At the same time, recent advances in large language models (LLMs) have spawned the study of *mechanistic interpretability*, which aims to identify granular model components like *circuits* that perform specific computations. In this work, we demonstrate the potential of circuit discovery techniques in NAR through two case studies, focusing on neuron-level circuits from algorithmic reasoning GNNs. Our study sheds new light on the process of circuit formation and pruning during training, as well as giving new insight into how GNNs trained to perform multiple tasks in parallel reuse circuit components for related tasks. Our GNN circuit discovery implementation, **M**echanistic **I**nterpretability for **N**eural **A**lgorithmic **R**easoning (MINAR), is available at `https://anonymous.4open.science/r/MINAR-ECD1/`.

1 INTRODUCTION

The recent surge in the capabilities of large language models (LLMs) has created a commensurate demand for novel interpretability methods suited to rigorous analysis of these models. At the same time, the discovery of phenomena like *grokking* (Power et al., 2022) has shown that rich algorithmic structure can arise via standard training methods. This confluence of interests has led to the emergence of *mechanistic interpretability* (Nanda et al., 2023), which attempts to explain model behavior using specific subsets of a model's internal computation. In several striking cases (Wang et al., 2023; Lindsey et al., 2025) this has allowed researchers to reduce seemingly complex behaviors to an interpretable sequence of mathematical operations.

At the same time, researchers have noticed that graph neural networks (GNNs) are capable of implementing classical algorithms as a result of the similarity between their message-passing structure and dynamic programming. This notion of "algorithmic alignment" (Xu et al., 2020) or "neural algorithmic reasoning" (Veličković et al., 2020), promises robust out-of-distribution generalization and substantial improvements in sample complexity. With an interest in discovering how models perform algorithmic tasks on the one hand and a class of models (GNNs) that are known to naturally implement traditional algorithms on the other, neural algorithmic reasoning creates a natural testbed for fine-grained mechanistic interpretability research.

In this work, we introduce **M**echanistic **I**nterpretability for **N**eural **A**lgorithmic **R**easoning (MINAR) to study circuit formation in algorithmic reasoning GNNs. We detail the construction of a (model) computation graph for a GNN and the precise formulation of standard attribution patching methods for GNNs. We also introduce a simple yet effective sensitivity analysis method which shows competitive performance compared to standard attribution patching (Section 4). To our knowledge, MINAR is the first attempt to apply automated circuit discovery to graph neural networks.

Through two case studies, we use circuit discovery to shed light on how GNNs form task-based circuits for algorithmic tasks. In the first case study (Section 5), we replicate recent work by Nerem et al. (2026) that trains a GNN to implement the classic Bellman-Ford algorithm. The ability of a graph neural network to implement Bellman-Ford is well-understood, presenting a strong ground truth to validate our method. Using MINAR, we extract a minimal circuit that captures the model's implementation of the Bellman-Ford algorithm. We also use MINAR to reveal how this circuit emerges during training by pruning a larger generalizable circuit, and we showcase how MINAR can

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.

be used to detect shortcut learning in a network trained to perform Bellman-Ford and breadth-first search (BFS) in parallel (Section 5.3). In the second case study (Section 6), we apply MINAR to a GNN trained to perform seven tasks from the CLRS-30 benchmark (Veličković et al., 2022) in parallel. Here, MINAR showcases how the model leverages the same circuit components to perform related tasks, as well as another instance of a circuit emerging well after model generalization.

2 Related Work

Circuit Discovery and Mechanistic Interpretability. Early mechanistic interpretability work, e.g., (Nanda et al., 2023; Zhong et al., 2023), combines manual examination of parameters with expert hypotheses to reverse-engineer the inner workings of a narrow trained model. However, the rise in language model capabilities has created an interest in applying such methods to much larger models, focusing on methods to partially or fully automate much of the mechanistic interpretability process. In particular, researchers have identified "circuits" that perform specific subtasks (Olah et al., 2020), and the process of identifying circuits is known as *circuit discovery*. A number of approaches for circuit discovery have been proposed, such as attribution patching (Syed et al., 2024; Hanna et al., 2024; Zhang et al., 2025) or pruning (Cao et al., 2021; Conmy et al., 2023; Yu et al., 2025). These methods formulate the model as a *computation graph* and attempt to find small subgraphs responsible for a particular behavior. To date, much of the circuit discovery literature has been focused on sub-tasks for language models such as indirect object identification (Wang et al., 2023).

Neural Algorithmic Reasoning with Graph Neural Networks. While mechanistic interpretability has revealed surprising structure in generalist LLMs, a different class of narrow models has also been shown to possess algorithmic "reasoning" capabilities: graph neural networks (GNNs). Broadly, GNNs operate via an iterative *message-passing* scheme: each node in the graph maintains an embedding, and iteratively aggregates the embeddings of its neighbors to update its own embedding. Xu et al. (2020) and Dudzik & Veličković (2022) point out that this aggregate-and-update flow resembles dynamic programming, hypothesizing that the *algorithmic alignment* of GNNs affords them an advantage in learning algorithmic tasks. For example, learning reachability via BFS and shortest paths via Bellman-Ford in parallel can help GNNs generalize on both tasks (Veličković et al., 2020). Veličković et al. (2022) later introduced the CLRS-30 benchmark to demonstrate this parallel reasoning ability across 30 algorithmic tasks inspired by the classic textbook *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein (Cormen et al., 2022), which has become a popular benchmark for NAR (Ibarz et al., 2022; Li et al., 2024).

3 Preliminaries

Before describing how circuit discovery can be applied to GNNs, we introduce the necessary background on circuit discovery (Section 3.1) and message-passing graph neural networks (Section 3.2).

3.1 Attribution Patching

Fundamental to circuit discovery is the *computation graph*, which represents the connections between individual model components (Olah et al., 2020; Syed et al., 2024; Zhang et al., 2025)[1]. In LLMs, these components represent large architectural modules like attention heads or MLPs. However, our work is more in line with the neuron-level circuits that Olah et al. (2020) identify in convolutional neural networks: each node in the computation graph corresponds to an individual neuron in the model, and each edge corresponds to a connection between neuron activations.

Definition 3.1 (Model Computation Graph)**.** Let Ψ be an arbitrary (perhaps non-graph) neural network with subcomponents ψ_i (e.g., neurons). The *model computation graph* of Ψ, denoted $G_c(\Psi)$, is the directed graph with vertices ψ_i and edges (i, j) if the output of ψ_i is part of the input to ψ_j.

Early circuit discovery work employs *activation patching*, which removes an edge from the computation graph and computes how this perturbation affects the model's predictions. (Meng et al., 2022;

[1] In GNN literature, some authors (Ying et al., 2019; You et al., 2021; He et al., 2025) use "computation graph" to refer to a GNN's layered message-passing, but in this paper "computation graph" will always refer to the model computation graph.

Conmy et al., 2023). Formally, given a loss $\mathcal{L}$ and input x, let (i, j) be an edge in the computation graph of a neural network Ψ, corresponding to the connection between two modules ψ_i and ψ_j. Let $\Psi^{\setminus(i,j)}$ be the neural network Ψ with computation edge (i, j) removed. Then (i, j) can be given an activation patching score

$$\textsc{ActPatch}(i, j) = \mathcal{L}\left(\Psi(x), \Psi^{\setminus(i,j)}(x)\right). \tag{1}$$

In the LLM setting, $\mathcal{L}$ may be the absolute logit difference for the "clean" output token (Syed et al., 2024) or the KL divergence between two token distributions (Conmy et al., 2023). Activation patching can be seen as a causal intervention on the model behavior. However, it becomes prohibitively expensive for larger computation graphs, as it requires a forward pass for each computation edge.

The goal of *attribution patching* is to *approximate* this difference in prediction (Hanna et al., 2024). Now, for a computation edge (i, j), we assign an attribution score of (i, j) for a prediction $\Psi(x)$ as follows: we first create a corrupted input x'. In the LLM setting, x' is typically a sentence where tokens of interest like names or years have been changed, but the overall sentence structure is preserved. Then, given the activations z_i, z'_i at ψ_i for inputs x, x', respectively and loss $\mathcal{L}(\Psi(x), \Psi(x'))$, the EAP score uses the gradient of $\mathcal{L}$ with respect to the input of ψ_j. That is,

$$\text{EAP}_{(i,j)}(x, x') = (z'_i - z_i)^{\perp} \frac{\partial}{\partial \psi_j} \mathcal{L}(\Psi(x), \Psi(x')). \tag{2}$$

Equation (2) comes from the first-order term in a Taylor expansion for the perturbation performed by activation patching, similar to the relationship between occlusion search and Input $\odot$ Gradient described by Ancona et al. (2018) in classical explainability. By perturbing the input and using the intermediate activations, EAP scores can be computed more efficiently than activation patching scores: EAP performs just two forward passes and one backward pass for each pair of inputs, rather than a forward pass for each computation edge (Syed et al., 2024).

Extending the analogy between circuit discovery and classical explainability, EAP-IG (Hanna et al., 2024) adapts EAP to use the integrated gradients method of Sundararajan et al. (2017), inheriting its advantages over Input $\odot$ Gradient. EAP-IG approximates an integral over the straight-line path between z_i and z'_i with a Riemann sum of m terms, essentially performing the EAP calculation m times (Hanna et al., 2024):

$$\text{EAP-IG}_{(i,j)}(x, x') = (z'_i - z_i)^{\perp} \frac{1}{m} \sum_{k=1}^{m} \frac{\partial}{\partial \psi_j} \mathcal{L}\left(\Psi(x), \Psi\left(x' + \frac{k}{m}(x - x')\right)\right). \tag{3}$$

3.2 Graph Neural Networks

Let Φ be a GNN operating on an attributed graph $G = (V, E, X, A)$, where V is the node set, E is the edge set, $X : V \to \mathbb{R}^p$ are the node features and $A : E \to \mathbb{R}^q$ are edge attributes. We briefly recall the message-passing scheme that underlies most popular GNNs. A GNN Φ of depth L maintains an embedding $\Phi_v^{(\ell)}$ for each node $v \in V$ and $\ell = 0, \ldots, L$, with layer 0 being the input node features $\Phi_v^{(0)} = X_v$. In each layer, $\Phi_v^{(\ell)}$ is updated according to the features of its 1-hop neighborhood $\mathcal{N}(v)$:

$$\Phi_v^{(\ell+1)} = f_{\text{Up}}^{(\ell)}\left(\Phi_v^{(\ell)}, \bigoplus_{u \in \mathcal{N}(v)} f_{\text{Agg}}^{(\ell)}(\Phi_u^{(\ell)}, A_{(u,v)})\right) \tag{4}$$

where $f_{\text{Agg}}^{(\ell)}$ and $f_{\text{Up}}^{(\ell)}$ are any functions and $\bigoplus$ denotes any permutation-invariant operation such as sum, mean, minimum, or maximum. We may denote the output of Φ at node v by Φ_v or $\Phi_v(G)$, if the input graph is ambiguous.

A key difference between GNNs and the models typically studied in circuit discovery is that a message-passing GNN (4) shares parameters across each node in the input graph, so any neuron ψ will necessarily have several activations on a single graph instance. Therefore, fully unrolling the computation graph would require a copy of ψ for every node in the input graph. This differs from circuit discovery in the LLM setting, e.g. (Wang et al., 2023), where the circuit can receive input directly from the model's residual stream and the circuit output is a single next-token prediction. Since our goal will be to extract a circuit which is shared across graphs, we will consider only a single copy of each neuron. We discuss in the next section how we accomodate the multiplicity of each neuron in the GNN computation graph.

4 CIRCUIT DISCOVERY FOR GRAPH NEURAL NETWORKS

We now turn to one of the central contributions of this work: how attribution patching can be applied to graph neural networks. We first describe our general approach to designing a corrupted input graph $G' = (V, E, X', A')$ from a clean input graph G. We use the same node set V and edge set E for G', but corrupt the features X' and A'. This way, G' remains structurally aligned with G, allowing for a one-to-one comparison of node and edge activations while ablating the actual computation. This is similar to the corrupted datasets of Wang et al. (2023), where sentences are kept structurally intact but the relevant semantic content is changed. (We discuss this choice further in Appendix D.3.) We refer to this probing dataset of clean graphs by $\mathcal{G}$ and the set of corresponding corrupted graphs by $\mathcal{G}'$.

To capture the role of each computation edge over each graph node involved in a forward pass, we pool attributions over input graph nodes. Because we consider node-level predictions in this work, we average attributions over each node in the probing set, i.e., we take the mean over $\mathcal{V} = \bigcup_{G \in \mathcal{G}} V(G)$. Given a GNN Φ and a loss $\mathcal{L}$ (e.g. nodewise MSE or cross-entropy), we define

$$\mathrm{EAP}_{(i,j)}(\mathcal{G}, \mathcal{G}') = \left| \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \left({z'}_i^v - z_i^v \right)^\perp \frac{\partial}{\partial \psi_i} \mathcal{L}(\Phi_v, \Phi_v) \right|, \tag{5}$$

where z_i^v is the activation of neuron ψ_i on node v and ${z'}_i^v$ is the corrupted activation of ψ_i on v. For EAP-IG, we implement the steps between the original and corrupted inputs by interpolating between all node and edge features, and again take the mean score over each node.

In addition to standard attribution patching, we also propose a sensitivity analysis on computation edges, taking the gradient of the loss between clean and corrupted outputs with respect to the weight:

$$\textsc{WeightGrad}(i, j) = \left| \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{\partial}{\partial W_{i,j}} \mathcal{L}(\Phi_v, \Phi_v) \right|. \tag{6}$$

WEIGHTGRAD is simpler and cheaper to compute than standard attribution patching. While it is normally ill-defined in the LLM setting, where computation edges represent large collections of weights, WEIGHTGRAD is a natural choice at the neuron level.

In Equations (5) and (6), we follow previous work in using the absolute value of each attribution score for downstream circuit discovery. In addition, since we focus on GNNs performing node-level predictions, we average over every *node* in the probing dataset. For GNNs performing graph-level prediction, it would be appropriate to pool the nodewise scores within each graph, and then average over graph-level scores, but this will not be relevant in this work.

Finally, we describe a simple baseline against which we can compare the methods described above. Since computation edges represent connections between individual neurons, we can simply use the absolute value of the weight $|W_{i,j}|$ between neurons φ_i and φ_j as a measure of importance. Once the model computation edges have been scored, we adopt the greedy Dijkstra-like algorithm from Hanna et al. (2024), which starts from the output neurons and greedily chooses high-scoring edges from the search frontier. The circuit is then pruned, ensuring that the resulting circuit does not contain isolated nodes. That is, all parentless nodes are model inputs and all childless nodes are model outputs. We describe alternative selection algorithms and provide an ablation study in Appendix A.

5 CASE STUDY: BELLMAN-FORD

In this section, we validate MINAR by studying GNNs trained to perform single-source shortest path distances, a common task for neural algorithmic reasoning research (Veličković et al., 2020; Xu et al., 2020; Nerem et al., 2026). In particular, both the algorithm itself and the mechanism by which a GNN can implement the algorithm are well-understood, providing an ideal case to compare recovered circuits to ground-truth.

MINAR recovers the minimal circuit in a GNN trained to compute single-source shortest path differences (Section 5.2). We also present a surprising observation that the parameters of the Bellman-Ford circuit generalize later than the full model, which we explain by using MINAR to reconstruct how the model is pruned during training. Section 5.3 extends this example by training on a second

task, breadth-first search (BFS), in parallel. Here, applying MINAR reveals that the model learned a shortcut for breadth-first-search based on the shortest path distance. Although we report here results from a single run, we observe the same behavior from different initializations (Appendix B.5) and experimental setups (Appendix B.4).

5.1 TRAINING

Throughout, we replicate the training setup of Nerem et al. (2026). We use a two-layer minimum-aggregated message-passing network (MinAggGNN), corresponding to two steps of Bellman-Ford. Recall that at the ℓ-th step of the Bellman-Ford algorithm, each node v updates its estimated distance $d_v^{(\ell)}$ to the source node s by

$$d_v^{(\ell+1)} \leftarrow \min_{u \in \mathcal{N}(v)} \{d_u^{(\ell)} + w_{u,v}\} \tag{7}$$

where $w_{u,v}$ is the weight of edge (u, v). Analogously, each layer of the MinAggGNN is given by taking a minimum as the aggregation option in (4).

$$\Phi_v^{(\ell+1)} = f_{\mathrm{Up}}^{(\ell)} \left(\Phi_v^{(\ell)}, \min_{u \in \mathcal{N}(v)} f_{\mathrm{Agg}}^{(\ell)}(\Phi_u^{(\ell)}, w_{u,v}) \right) \tag{8}$$

where $w_{u,v}$ is given as an edge feature for (u, v). We implement $f_{\mathrm{Agg}}^{(\ell)}$ and $f_{\mathrm{Up}}^{(\ell)}$ as two-layer MLPs, denoting by $f_{\mathrm{Agg}}^{(\ell;k;n)}$ the n-th neuron in the k-th linear layer of $f_{\mathrm{Agg}}^{(\ell)}$ (and similarly for $f_{\mathrm{Up}}^{(\ell)}$). By using minimum aggregation, the GNN only needs to learn a simple inner function to emulate Equation (7). By training with L_1 sparsity regularization and a small curated training set, Nerem et al. (2026) showed that if a GNN achieves a sufficiently low regularized training loss, then it must implement the Bellman-Ford algorithm. This type of regularization has been shown to be necessary for learning guarantees in a broader class of algorithmic tasks (Wittig et al., 2026), and we study how circuits emerge in this setting.

The training set consists of few small graphs of size up to 6 nodes. To ensure that the test set captures out-of-distribution generalization, the test set consists of hundreds of randomly generated graphs up to 200 nodes. Nodes are given input features that emulate the standard initialization of the Bellman-Ford algorithm from some source node s, with $x_s^{\mathrm{SP}} = 0$ and $x_v^{\mathrm{SP}} = B$ otherwise. Here B is a large number greater than any path length, which we take to be $B = 1000$. We discuss the dataset in greater detail in Appendix D.1. While we use mean squared error (MSE) during training, we evaluate on the test set $\mathcal{G}_{\text{test}}$ using a multiplicative test loss $\mathcal{L}_{\text{Mult}}$ which is not sensitive to the scale of the edge weights. We compute $\mathcal{L}_{\text{Mult}}$ over the set $\mathcal{N}^{(L)}(s)$ of nodes at most L steps from s:

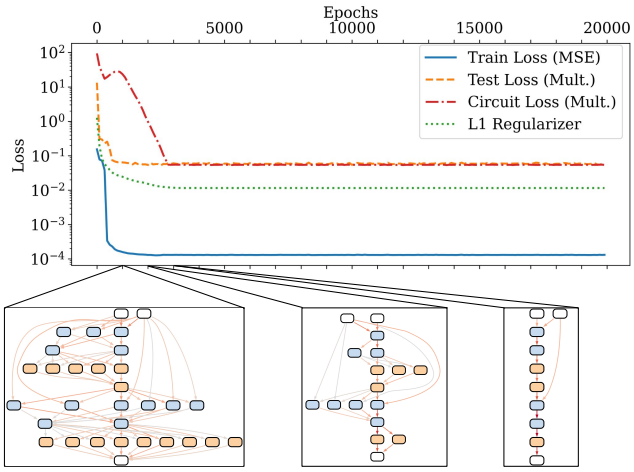
$$\mathcal{L}_{\text{Mult}} = \frac{1}{|\mathcal{G}_{\text{test}}|} \sum_{G \in \mathcal{G}_{\text{test}}} \sum_{v \in \mathcal{N}^{(L)}(s)} \left| 1 - \frac{y_v}{\Phi_v(G)} \right| \tag{9}$$

where y_v is the true label for each reachable node. The MinAggGNN reaches a train loss of $\mathcal{L}_{\mathrm{MSE}} = 0.0001$ and a test loss of $\mathcal{L}_{\text{Mult}} = 0.0578$ (Figure 1a).

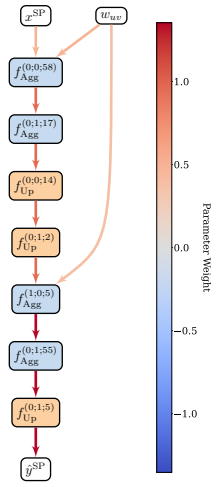
5.2 THE BELLMAN-FORD CIRCUIT

For circuit discovery, we corrupt the input data by zero-ablating the edge weights to $A_{u,v} = 0$ and the input features to $x_v^{\mathrm{SP}'} = 0$. Using this corruption with MSE, we use each scoring method in Section 4 to identify a circuit in the 18240-edge computation graph. All methods except WEIGHTGRAD are able to identify the the theoretically optimal circuit of 10 parameters (Figure 1b) predicted by Nerem et al. (2026). (A comparison of scoring methods is given in Appendix B.2, where we see that WEIGHTGRAD requires 12 parameters to find a sufficient circuit; we see this trend across all seeded runs, as we describe in Appendix B.)

Moreover, by applying circuit discovery to model checkpoints early in training, we can see how the model prunes an initial solution (which generalizes well but is not concise) into the final minimal circuit (Figure 1a). The final circuit is sufficient to replicate the model's Bellman-Ford implementation: taken as a subnetwork of the full model, the circuit achieves $\mathcal{L}_{\text{Mult}} = 0.0545$ on the test set, matching the high performance of the full model. (In fact, the circuit achieves a slightly lower loss overall.) A detailed verification that the circuit implements Bellman-Ford is provided in Appendix B.1. It is also



(a) Train loss (MSE), Test Loss ($\mathcal{L}_{\text{Mult}}$) Circuit test loss ($\mathcal{L}_{\text{Mult}}$), and L_1 regularization term for Bellman-Ford MinAggGNN (top). Circuits (below) are extracted from epochs $t = 1000$ (left), $t = 2000$ (middle), and $t = 3000$ (right).



(b) Final Bellman-Ford circuit, with circuit edges colored by corresponding model weights.

Figure 1: Training curves and circuit formation (Figure 1a, left) with final Bellman-Ford circuit (Figure 1b, right). A generalizable circuit forms early in training, but L_1 regularization prunes until reaching the minimal circuit. Each circuit in Figure 1a contains the final circuit as a subcircuit.

necessary: when the circuit is removed, the resulting loss is $\mathcal{L}_{\text{Mult}} = 21870.6836$. In this example, MINAR successfully recovers the minimal circuit that captures the full model behavior.

The emergence and pruning of the Bellman-Ford circuit resembles observations from Elhage et al. (2022); Nanda et al. (2023); Zhong et al. (2023); Li et al. (2025) that once a neural circuit emerges during training, regularization can prune extraneous parameters from the model. This has been put forward as an explanation for the "grokking" phenomenon, where models learn general solutions well after achieving perfect performance on the training set. However, what we observe has a key qualitative difference: rather than the model learning several heuristics and rules before pruning down to a generalizable implementation (removing so-called "vestigial circuits" (Li et al., 2025)), we find that the model achieves a generalizable solution relatively early, perhaps due to the algorithmic alignment of the architecture. Then, instead of having to prune vestigial circuits, the pruning we observe simply forces the model to make the circuit more concise. We provide further discussion of this pruning behavior in Appendix B.3, and analyze a striking example in Appendix B.4 where the model is forced to partially learn the minimum operation within the update MLP.

5.3 Shortcut Learning: Using Bellman-Ford for BFS

We extend the Bellman-Ford example by introducing a second task: predicting reachability from a source node by BFS. Veličković et al. (2020) note that Bellman-Ford and BFS traverse a graph in the same manner, and show that training these tasks in parallel can benefit performance on both. Here, MINAR reveals surprising behavior in this two-task setup: the model learns a shortcut for BFS based entirely on the shortest path distance.

Both tasks share the same training set, but with additional node features and labels to support the reachability task. As with Bellman-Ford, the model's two layers correspond to two steps of BFS. In addition to the Bellman-Ford encoding, each node is given an initial feature for BFS that one-hot encodes the same source s (i.e. $x_s^{\text{BFS}} = 1$ and $x_v^{\text{BFS}} = 0$ otherwise), and we corrupt the dataset by zero-ablation. The parallel GNN receives as input the concatenation $(x_v^{\text{SP}}, x_v^{\text{BFS}})$. For the test set, we additionally include a number of balanced tree graphs to provide more graphs with unreachable nodes. We train using MSE for the shortest path task and binary cross-entropy (BCE) for the reachability task. The model achieves a test loss of $\mathcal{L}_{\text{Mult}} = 0.0519$ on shortest paths and a test accuracy of 1.00 on reachability. (More details are given in Appendix E and train curves in Figure 10, Appendix B).

We again use EAP-IG with $m = 20$ steps to identify a circuit (Figure 2). The circuit, which performs both tasks, contains 11 edges (out of 18432) that achieves $\mathcal{L}_{\text{Mult}} = 0.0499$ on shortest paths and a reachability accuracy of 0.9831, showing that the circuit is sufficient. It is also necessary: removing it results in $\mathcal{L}_{\text{Mult}} = 58038.1680$ and a test accuracy of 0.8198 (the accuracy when the model classifies every node as reachable).

Upon inspection, it is clear that the circuit does not perform BFS independently—it is almost identical to the Bellman-Ford circuit, differing only by an edge to the BFS output neuron. While concerns about shortcut learning often center around poor out-of-distribution generalization (Geirhos et al., 2020), MINAR exposes shortcut learning which is invisible even to our large out-of-distribution test set. Manual inspection of the parameters represented in the BFS circuit reveals that the reachability prediction is given by $\hat{y}^{\text{SP}} \approx \hat{y}^{\text{BFS}} - 9.0402$. This implies that the model has learned a shortcut: predict "reachable" if and only if a node's estimated shortest path distance is less than about 9. (For a full detailed analysis, see Appendix B.1.)

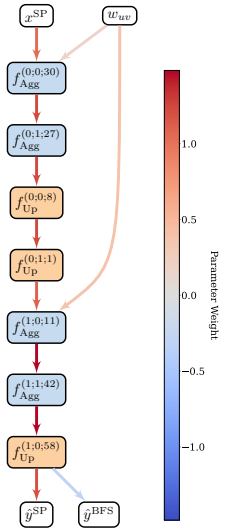


Figure 2: Bellman-Ford and BFS circuit. The circuit bypasses BFS encodings and is almost identical to the Bellman-Ford-only circuit.

6 Case Study: SALSA-CLRS

In this section we apply circuit discovery to a collection of graph-based reasoning tasks taken from the popular CLRS-30 benchmark (Veličković et al., 2022), which has been used to study multi-task reasoning ability in GNNs (Ibarz et al., 2022; Li et al., 2024). We offer a circuit perspective on this phenomenon: the model learns distinct but overlapping circuits which are responsible for each task.

6.1 Architecture and Training

We use the PyTorch Geometric implementations of breadth-first search (BFS), depth-first search (DFS), Dijkstra's shortest path algorithm, and Prim's minimum spanning tree (MST) algorithm from SALSA-CLRS Minder et al. (2023). We also include implementations of Bellman-Ford, Articulation Points, and Bridges from CLRS-30 in the SALSA-CLRS framework. These seven tasks represent the subset of CLRS tasks which can be represented on sparse graphs. [2]

Following the Encode-Process-Decode setup described in (Veličković et al., 2022; Minder et al., 2023), each task has its own linear encoder and decoder for node and edge features. In addition, some tasks require the network to provide a more complex prediction in the form of a node "pointer," e.g. from a node to its BFS predecessor. The processor is a graph neural network Φ shared across all tasks, which is applied recurrently to the encoded features. We use a 2-layer Graph Isomorphism Network with Edge Features (GINE) (Hu et al., 2020), modified from the Graph Isomorphism Network in (Xu et al., 2019). We follow earlier NAR work by using maximum aggregation (Veličković et al., 2020; Veličković et al., 2022; Ibarz et al., 2022), so each layer is given by

$$\Phi_v^{(\ell+1)} = f^{(\ell)}\left(\Phi_v^{(\ell)} + \max_{u \in \mathcal{N}(v)} \text{ReLU}(\Phi_u^{(\ell)} + W_e^{(\ell)} e_{u,v})\right), \tag{10}$$

where $f^{(\ell)}$ is a two-layer MLP and $W_e^{(\ell)}$ is a linear map which maps the edge feature $e_{u,v}$ to the same dimension as the node embedding space. Training details are provided in Appendix E. Because the SALSA-CLRS experiment is more computationally expensive, we are limited to 3 seeded runs. We discuss here results for one run, and provide results from the additional two runs in Appendix C. (The exact circuit discovery results are difficult to aggregate between runs, as circuit pruning means we end up sampling circuits at slightly different sparsity with each seed.)

[2] The remaining CLRS-30 tasks require complete graphs to represent problem instances, as CLRS-30's hints allow arbitrary node connections. We restrict ourselves to tasks where hints are restricted to edges which already exist in the graph. For further discussion, see Appendix D.2.

Table 1: Validation and out-of-distribution test performance of maximum-aggregated GINE on breadth-first search (BFS), depth-first search (DFS), Dijkstra's algorithm, Prim's minimum spanning tree (MST), Bellman-Ford (B-F), Articulation Points (Art. Points), and Bridges.

	Node Accuracy (%)					Node F_1	Edge F_1
Algorithm	BFS	DFS	Dijkstra	Prim's MST	B-F	Art. Points	Bridges
Validation	99.86	76.50	90.10	82.72	90.68	65.15	67.76
Test	96.40	21.90	67.75	50.42	65.23	23.02	26.51

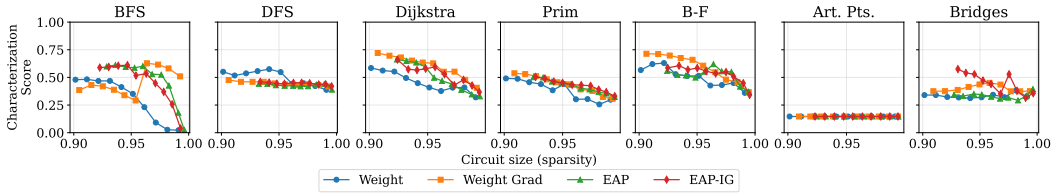


Figure 3: Characterization score (higher is better) of circuits computed using Weight, WEIGHTGRAD, EAP, and EAP-IG ($m = 20$) for SALSA-CLRS GNN.

Table 1 shows the validation accuracy and (out-of-distribution) testing accuracy of the trained model, showing varying degrees of success. The model generalizes well on BFS, achieving nearly perfect validation accuracy and maintaining high performance out-of-distribution. On Dijkstra's algorithm, Prim's algorithm, and Bellman-Ford, the model learns well in-distribution, but struggles to generalize to larger graphs. On DFS, Articulation Points, and Bridges, the model has difficulty learning even the training and validation distribution, and has very poor generalization performance.

6.2 CIRCUIT DISCOVERY AND ANALYSIS

To perform circuit discovery, we corrupt a new probing dataset of unseen graphs drawn from the validation distribution. We zero-ablate every input feature, including node positions on all graphs; source nodes for BFS, DFS, Dijkstra's, Prim's, and Bellman-Ford; and edge weights on Dijkstra's, Prim's, and Bellman-Ford. Because the processor is responsible for the algorithmic reasoning steps, we focus on identifying circuits within the processor GNN using the same loss that was used in training. A more detailed discussion can be found in Appendix D.2.

We use MINAR to identify circuits in the processor's computation graph of 99712 edges using WEIGHTGRAD, EAP, and EAP-IG ($m = 20$), with simple model weights as a baseline. We evaluate circuits using the *characterization score* from Amara et al. (2022), which balances sufficient and necessary explanations. Figure 3 shows the characterization score of circuits at different levels of model sparsity. (A precise definition of fidelity scores and the individual fidelity scores for each circuit are given in Appendix C.) We can see that WEIGHTGRAD, EAP, and EAP-IG generally outpace the weight baseline in finding explanatory circuits, particularly for well-learned tasks. Despite its simplicity, WEIGHTGRAD performs quite well, matching or even exceeding EAP and EAP-IG on several tasks. We also note that the sudden change in characterization score for WEIGHTGRAD is a result of the circuit suddenly failing to be necessary; it in fact remains the most faithful (i.e. sufficient) circuit compared to the EAP and EAP-IG circuits (Figure 27, Appendix C).

BFS Circuit Formation Perhaps unsurprisingly, the success of attribution patching seems to roughly match the model's generalization performance. As we saw in Section 5, a circuit may not be sufficient to replicate the model performance until well after the model generalizes. We observe a similar phenomenon with the BFS circuit here: when evaluating the most faithful Bellman-Ford circuit at $> 90\%$ model sparsity on validation graphs, we can see two distinct phase changes (Figure 4). Prior to $t = 48$ epochs, the circuit significantly underperforms the full model. At around $t = 48$, the circuit's maximum performance increases dramatically, but continues to be unstable. Finally, at $t = 69$, the circuit stabilizes, maintaining its performance for the rest of training.

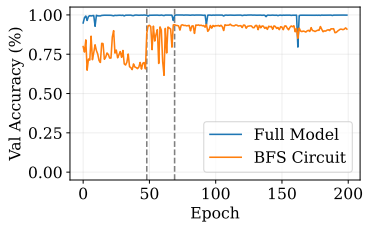


Figure 4: Validation accuracy of BFS circuit vs. full GNN.

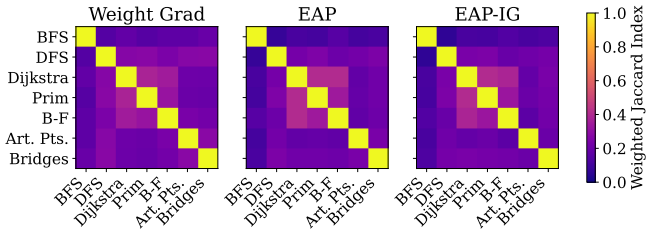


Figure 5: Weighted Jaccard Similarity of circuits with WEIGHTGRAD, EAP, EAP-IG at $\sim 90\%$ model sparsity.

Circuit Overlap We can also study how the learned circuits share parameters. Figure 5 displays the overlap of circuits identified with WEIGHTGRAD, EAP, and EAP-IG. (We do not plot the overlap of circuits found with Weight, as it is invariant to the task and therefore all the circuits are identical.) We measure circuit overlap by the weighted Jaccard similarity between the circuit edge sets, where edges are weighted by the absolute value of their corresponding model weights. (Additional overlap comparisons for different scoring methods can be found in Appendix C.) We highlight two observations from the circuit overlap: first, across patching methods BFS maintains separation from the other circuits, being the most well-defined. Second, we can see significant overlap between Dijkstra's algorithm, Prim's algorithm, and Bellman-Ford; these three are naturally grouped together, as they are the three tasks that use edge weights. This circuit overlapping phenomenon provides a potential explanation for transfer learning observations like (Veličković et al., 2020), where training on similar tasks in parallel helps GNNs perform on each individual task.

7 DISCUSSION

In this work, we use MINAR to provide the first study of automated circuit discovery for GNNs. In two case studies, we use circuit discovery to extract faithful circuits from GNNs trained for algorithmic tasks, and we examine how circuits are formed and pruned during training in NAR settings. Our observations are similar to those in prior work that regularization can prune extraneous parameters from a model after a neural circuit emerges during training, (Elhage et al., 2022; Nanda et al., 2023; Zhong et al., 2023; Li et al., 2025). However, the pruning that we observe differs from other examples: unlike pruning "vestigial circuits" (Li et al., 2025) which implement heuristics or memorize training data, we observe that the L_1 regularization used in the NAR settings of Nerem et al. (2026); Wittig et al. (2026) refines a generalizing circuit to be more concise. We also show how GNNs performing multiple algorithmic tasks in parallel share circuit components for similar tasks—in the most extreme case exposing severe shortcut learning. We hope that by bringing the emerging field of mechanistic interpretability to the study of neural algorithmic reasoning, MINAR provides a useful tool to study trained NAR models in detail.

Limitations and Future Work Our work is focused on two examples from the field of neural algorithmic reasoning. While these synthetic tasks are useful for validating interpretability methods, as the desired model behavior corresponds to a well-known ground truth, this limits the scope of claims we can make in more general settings. We have also observed in our experiments that circuit discovery is especially difficult when a model is not well-trained. While we believe this is because circuit formation and model training are tightly related, this is a limitation of circuit discovery and it is possible that other trends appear in poorly trained models. Since the ultimate goal of both NAR and mechanistic interpretability is to leverage and discover algorithmic knowledge from more difficult and complex domain problems, future work could combine insights from circuit discovery with other tools from interpretability such as concept-based neuron explanations (Bau et al., 2017; Mu & Andreas, 2020; Hernandez et al., 2021; Oikarinen & Weng, 2023) or sparse autoencoders (Huben et al., 2024) in order to gain a more complete understanding of GNNs in application domains.

AI USE STATEMENT

In this work, we used generative AI tools for coding assistance, including implementing methods, writing code to plot results, and writing code to generate synthetic datasets. We have not used generative AI tools to develop conceptual frameworks, propose hypotheses, design research methodology, or analyze results, and the rest of the required disclosure tasks are not applicable to this work. The lead author verified and tested LLM-generated code. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide code for generating data and implementing methods in an anonymous repository. Detailed descriptions of methods are provided in Section 4 and Appendix A, data generation is described in Appendix D, and we describe hyperparameters in detail in Appendix E.

REFERENCES

Kenza Amara, Zhitao Ying, Zitao Zhang, Zhichao Han, Yang Zhao, Yinan Shan, Ulrik Brandes, Sebastian Schemm, and Ce Zhang. GraphFramEx: Towards systematic evaluation of explainability methods for graph neural networks. In Bastian Rieck and Razvan Pascanu (eds.), *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pp. 44:1–44:23. PMLR, 09–12 Dec 2022. URL https://proceedings.mlr.press/v198/amara22a.html.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Sy21R9JAW.

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pp. 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150464. URL https://doi.org/10.1145/1150402.1150464.

Steven Cao, Victor Sanh, and Alexander Rush. Low-complexity probing via finding subnetworks. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 960–966, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.74. URL https://aclanthology.org/2021.naacl-main.74/.

Kenzo Clauw, Daniele Marinazzo, and Sebastiano Stramaglia. Information-theoretic progress measures reveal grokking is an emergent phase transition. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL https://openreview.net/forum?id=Q4NH6hEPIX.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 16318–16352. Curran Associates, Inc.,

2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2022.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.581. URL https://aclanthology.org/2022.acl-long.581/.

Branton DeMoss, Silvia Sapora, Jakob Foerster, Nick Hawes, and Ingmar Posner. The complexity dynamics of grokking. *Physica D: Nonlinear Phenomena*, 482:134859, 2025. ISSN 0167-2789. doi: https://doi.org/10.1016/j.physd.2025.134859. URL https://www.sciencedirect.com/science/article/pii/S0167278925003367.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271, 1959. doi: 10.1007/BF01386390.

Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20635–20647. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8248b1ded388fcdbbd121bcdfea3068c-Paper-Conference.pdf.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. *ArXiv*, abs/1903.02428, 2019. URL https://api.semanticscholar.org/CorpusID:70349949.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016. URL http://arxiv.org/abs/1510.00149.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL https://openreview.net/forum?id=grXgesr5dT.

Jesse He, Akbar Rafiey, Gal Mishne, and Yusu Wang. Explaining GNN explanations with edge gradients. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 884–895, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3736947. URL https://doi.org/10.1145/3711896.3736947.

Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH.

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.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=FebadKZf6Gd.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vt5mnLVIVo.

Hefei Li, Chao Peng, Chenyang Xu, and Zhengfeng Yang. Open-book neural algorithmic reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6HO33urpaI.

Victoria R Li, Jenny Kaufmann, Martin Wattenberg, David Alvarez-Melis, and Naomi Saphra. Can interpretation predict behavior on unseen data? In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=fzw0FT8Clf.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: an effective theory of representation learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=zDiHoIWa0q1.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Neil Rohit Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: grokking modular arithmetic via average gradient outer product. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=36hVB7DEB0.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Julian Minder, Florian Grötschla, Joël Mathys, and Roger Wattenhofer. SALSA-CLRS: A sparse and scalable benchmark for algorithmic reasoning. In *The Second Learning on Graphs Conference*, 2023. URL https://openreview.net/forum?id=PRapGjDGFQ.

Edward F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pp. 285–292, Cambridge, MA, USA, 1959. Harvard University Press.

Jesse Mu and Jacob Andreas. Compositional explanations of neurons. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17153–17163. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c74956ffb38ba48ed6ce977af6727275-Paper.pdf.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Robert R. Nerem, Samantha Chen, Sanjoy Dasgupta, and Yusu Wang. Graph neural networks extrapolate out-of-distribution for shortest paths. In Steve Hanneke and Tor Lattimore (eds.), *Proceedings of Thirty Ninth Conference on Learning Theory*, volume 336 of *Proceedings of Machine Learning Research*, pp. 5273–5331. PMLR, 29 Jun–03 Jul 2026. URL https://proceedings.mlr.press/v336/nerem26a.html.

Tuomas Oikarinen and Tsui-Wei Weng. CLIP-dissect: Automatic description of neuron representations in deep vision networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iPWiwWHc1V.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. URL https://distill.pub/2020/circuits/zoom-in.

Abhishek Panigrahi, Bingbin Liu, Sadhika Malladi, Andrej Risteski, and Surbhi Goel. Progressive distillation induces an implicit curriculum. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=wPMRwmytZe.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Robert C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. doi: 10.1002/j.1538-7305.1957.tb01515.x.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 3319–3328. JMLR.org, 2017.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen (eds.), *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 407–416, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.blackboxnlp-1.25. URL https://aclanthology.org/2024.blackboxnlp-1.25/.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 22084–22102. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/velickovic22a.html.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

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.

Solveig Wittig, Antonis Vasileiou, Robert R Nerem, Timo Stoll, Floris Geerts, Yusu Wang, and Christopher Morris. Which algorithms can graph neural networks learn? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=GnmuZIlvxw.

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.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/d80b7040b773199015de6d3b4293c8ff-Paper.pdf.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10737–10745, 2021. doi: 10.1609/aaai.v35i12.17283.

Lei Yu, Jingcheng Niu, Zining Zhu, Xi Chen, and Gerald Penn. Sheaf discovery with joint computation graph pruning and flexible granularity. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8822–8837, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.446. URL https://aclanthology.org/2025.emnlp-main.446/.

Lin Zhang, Wenshuo Dong, Zhuoran Zhang, Shu Yang, Lijie Hu, Ninghao Liu, Pan Zhou, and Di Wang. EAP-GP: Mitigating saturation effect in gradient-based automated circuit identification. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=lGyXq0LOeQ.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S5wmbQc1We.

A Circuit Selection Algorithms

In most experiments, we adopt the greedy algorithm of Hanna et al. (2024), which begins at the model's output logits and greedily adds edges whose children are in C and whose parents are not in C Algorithm 1. Because circuit discovery frequently stipulates that C should not have isolated nodes

Algorithm 1 Greedy circuit construction algorithm

Input: G_c, `Scores`, K
Initialize circuit $C \leftarrow$ `Outputs`
for $k < K$ **do**
  `Frontier` $\leftarrow \{(i, j) \in G_c : j \in C \text{ and } i \notin C\}$
  $(i^*, j^*) \leftarrow \arg\max_{\texttt{Frontier}}$ `Scores`
  $C \leftarrow C \cup (i^*, j^*)$
end for
return C

(i.e. any parentless nodes are input neurons and childless nodes are output logits), we recursively prune isolated nodes from the discovered circuit.

We also briefly present an algorithm that takes advantage of the fact that the computation graph is directed and acyclic, and therefore supports an efficient longest-path algorithm. One can find the longest path in a DAG in $O(|V(G_c)| + |E(G_c)|)$ time, the same runtime as constructing the model computation graph G_c itself. Moreover, finding the longest path which contains a specific edge (i, j) reduces to two instances of the longest path algorithm on a subgraph of G_c, so it also takes $O(|V(G_c)| + |E(G_c)|)$ time. Algorithm 2 describes this circuit discovery algorithm. Once scores are computed and the computation graph is constructed, topologically sorting G_c takes time $O(|V(G_c)| + |E(G_c)|)$, and sorting the edges by attribution score takes time $O(|E(G_c)| \log(|E(G_c)|))$. After these preprocessing steps, finding a circuit that includes the top K edges takes time $O(K(|V(G_c)| + |E(G_c)|))$.

Algorithm 2 Longest path circuit construction algorithm

Input: G_c, `Scores`, K
TopologicalSort(G_c)
SortDescending(`Scores`)
Initialize circuit C
$k \leftarrow 0$
while $k < K$ and `Scores` $\neq \varnothing$ **do**
  $(i, j) \leftarrow$ `Scores`.Pop()
  if $(i, j) \notin C$ **then**
    $C \leftarrow C \cup$ LongestPathWithEdge($G_c, (i, j)$)
    $k \leftarrow k + 1$
  end if
end while
return C

By constructing the circuit from complete input-output paths, Algorithm 2 guarantees that the identified circuit is a connected subgraph of the computation graph without isolated nodes. This ensures that none of the top K edges are pruned from the circuit. However, this comes at the cost of potentially including unnecessary edges. This is in contrast to naive Top-K selection followed by pruning, which is also common (Syed et al., 2024; Zhang et al., 2025). In some respects, we can position Top-K selection on one extreme and Algorithm 2 on the other, with Greedy lying somewhere in between.

When circuits are particularly well-formed and concise, as in Section 5, we find that these methods perform comparably, with Algorithm 2 typically requiring fewer iterations to converge on a circuit. However, for the model in Section 6 where circuits are substantially less well-defined and thus difficult to identify, we find that the unnecessary edges introduced by Algorithm 2 generally hinder the efficiency of circuit discovery compared to Greedy and Top-K. (Figures 6, 7, 8 and 9)

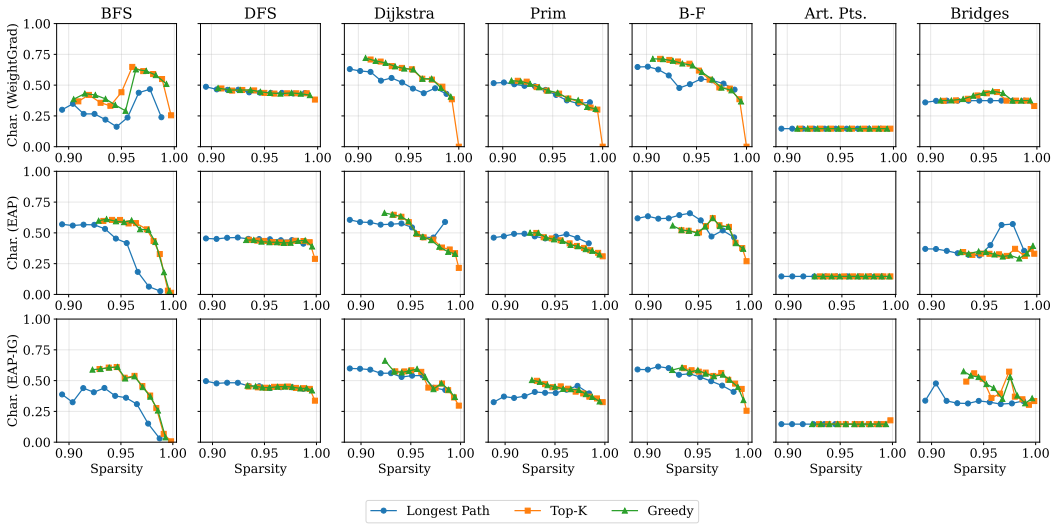


Figure 6: Characterization scores of circuits found with Longest Path, Top-K, and Greedy.

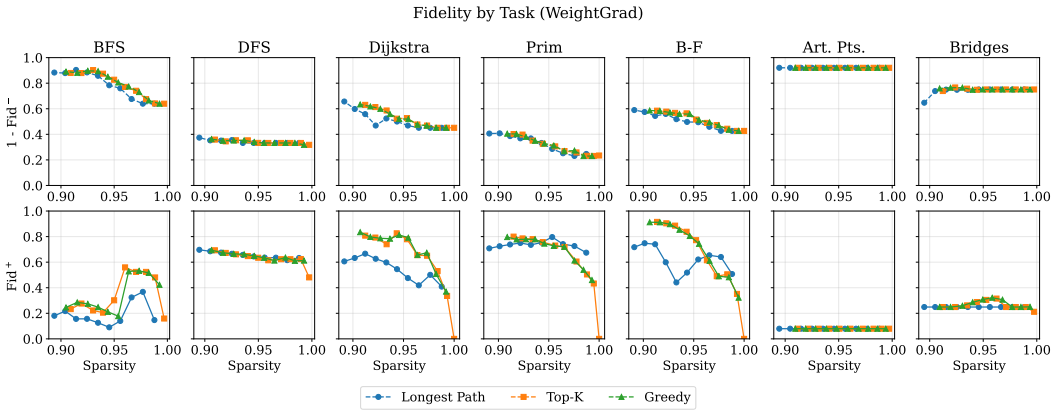


Figure 7: Fidelity of circuits found with Longest Path, Top-K, and Greedy using WEIGHTGRAD.

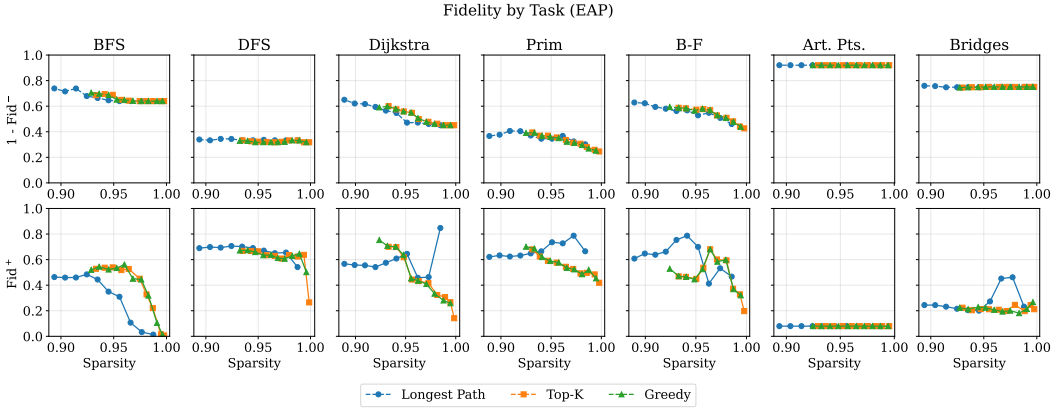


Figure 8: Fidelity of circuits found with Longest Path, Top-K, and Greedy using EAP.

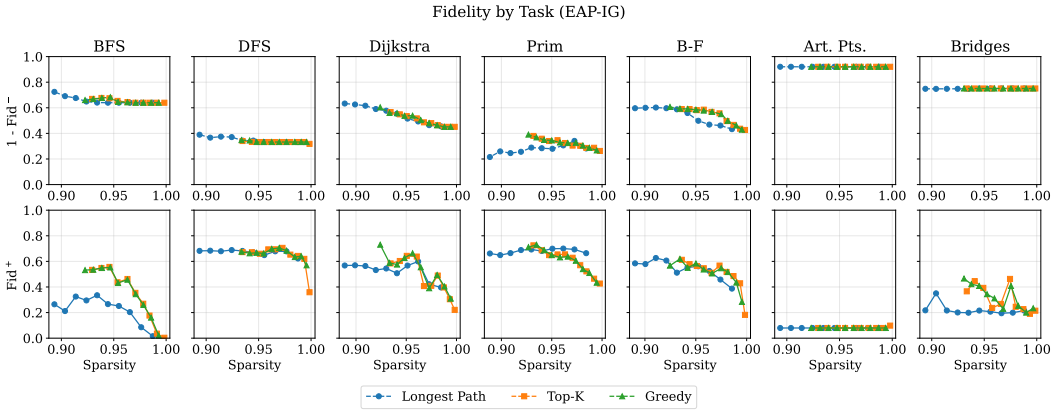


Figure 9: Fidelity of circuits found with Longest Path, Top-K, and Greedy using EAP.

B Additional Results for Bellman-Ford

This appendix contains training curves for the Bellman-Ford and BFS GNN (Figure 10), a full analysis of the Bellman-Ford circuit and shortcut (Appendix B.1), additional results from experiments in Section 5 (Appendix B.3), as well as relevant plots from runs with different training data (Appendix B.4) and seeds (Appendix B.5).

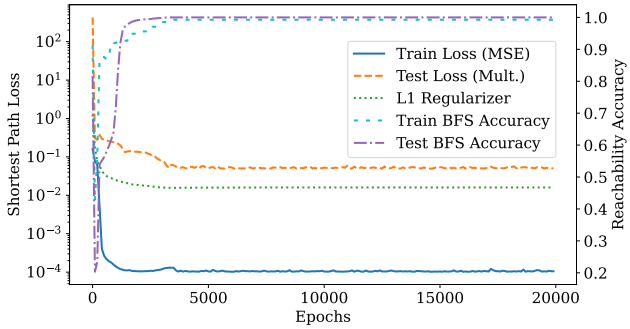


Figure 10: Train MSE, test $\mathcal{L}_{\text{Mult}}$, L_1 regularizer, and train and test accuracy for parallel Bellman-Ford and BFS GNN.

B.1 Mathematical Analysis of Circuits

B.1.1 The Bellman-Ford Circuit Implements Bellman-Ford

Here we will give an explicit analysis of every parameter in the Bellman-Ford circuit (Figure 1b). Recall that the network performs two steps of the Bellman-Ford algorithm. In the first layer, we have for each edge (u, v) (including self-loops) that

$$f_{\text{Agg}}^{(0;0;58)}(u,v) = \text{ReLU}\left([0.4771 \quad 0.4771]\begin{bmatrix} x^{\text{SP}_v} \\ w_{u,v} \end{bmatrix} + 0.0003\right) \tag{11}$$

$$f_{\text{Agg}}^{(0;1;17)}(u,v) = \text{ReLU}\left(0.9556 \cdot f_{\text{Agg}}^{(0;0;58)}(u,v) + 0.0002\right) \tag{12}$$

Note that $x^{\text{SP}_v} \geq 0$ and $w_{u,v} \geq 0$, so we may ignore the ReLU. Thus taken together, this yields

$$f_{\text{Agg}}^{(0;1;17)}(u,v) = [0.4559 \quad 0.4559]\begin{bmatrix} x^{\text{SP}_v} \\ w_{u,v} \end{bmatrix} + 0.0005. \tag{13}$$

Here, the minimum aggregation is applied over all $u \sim v$. Note that this includes v itself, so we have

$$f_{\text{Up}}^{(0;0;14)}(v) = \text{ReLU}\left(0.9557 \cdot \min_{u \sim v}\{f_{\text{Agg}}^{(0;1;17)}(u,v)\} - 0.0001\right) \tag{14}$$

$$f_{\text{Up}}^{(0;1;2)}(v) = \text{ReLU}\left(0.9559 \cdot f_{\text{Up}}^{(0;0;14)}(v) + 0.0008\right) \tag{15}$$

and hence

$$f_{\text{Up}}^{(0;1;2)}(v) = 0.9136 \cdot \min_{u \sim v}\{f_{\text{Agg}}^{(0;1;17)}(u,v)\} + 0.0007 \tag{16}$$

$$= 0.4165 \cdot \min_{u \sim v}\left\{x^{\text{SP}_v} + w_{u,v}\right\} + 0.0012. \tag{17}$$

Recalling that the output $d^{(1)}$ of one iteration of Bellman-Ford is given by $\min_{u \sim v}\left\{x^{\text{SP}_v} + w_{u,v}\right\}$, we see that the output of the first MinAggGNN layer in the circuit is simply a scaled implementation (with some small error):

$$\Phi^{(1)}(v) = 0.4165 \cdot d^{(1)} + 0.0012. \tag{18}$$

A similar analysis shows that

$$\hat{y}_v^{\text{SP}} = \Phi^{(2)}(v) = \min_{u \sim v}\left\{2.3953 \cdot \Phi^{(1)}(u) + w_{u,v}\right\} + 0.0022 \tag{19}$$

$$= \min_{u \sim v}\left\{0.9977 \cdot d^{(1)}(u) + w_{u,v}\right\} + 0.0050 \tag{20}$$

$$\approx d^{(2)}(u) \tag{21}$$

as desired.

B.1.2 The Bellman-Ford and BFS Shortcut

Since the structure of the parallel Bellman-Ford and BFS circuit (Figure 2) is nearly identical to the Bellman-Ford only circuit (Figure 1b), we focus our analysis on the shortcut in the final linear layer. Since the second linear layer of $f_{\text{Up}}^{(2)}$ gives the node-level output, we have

$$\hat{y}^{\text{BFS}} = (-.2927) f_{\text{Up}}^{(2;58)} + 2.6463. \tag{22}$$

But the neuron $f_{\text{Up}}^{(2;58)}$ also controls the shortest path prediction, as

$$\hat{y}^{\text{SP}} = (1.1934) f_{\text{Up}}^{(2;58)} + 0.0001. \tag{23}$$

We can write down the explicit relationship between the two predictions: disregarding the bias term for $\hat{y}^{\text{SP}}$ as negligible, these two equations together yield

$$\hat{y}^{\text{BFS}} \approx (-0.2453)\hat{y}^{\text{SP}} + 2.2175 \tag{24}$$

or

$$\hat{y}^{\text{SP}} \approx \hat{y}^{\text{BFS}} - 9.0402. \tag{25}$$

Since the task is binary classification, the model's prediction is "reachable" if $\hat{y}^{\text{BFS}} > 0$ and "unreachable" otherwise. Rearranging, this equation reveals that the model predicts "reachable" if and only if the shortest path prediction is less than about 9:

B.2 Score Comparison for Bellman-Ford

We compare in Figure 11 different scoring methods for the Bellman-Ford GNN in Section 5.2. Only WeightGrad fails to find the optimal 10-edge circuit, instead requiring 12 edges.

B.3 Circuit Sparsification

Here we discuss how the circuit in Section 5.2 (Figure 1b) emerges during training by extracting circuits from earlier epochs. We identify sufficient circuits at epochs 1000 and 2000. The final circuit appears by epoch 3000. This mirrors observations in LLMs where features may not align with individual neurons, but limiting capacity (either architecturally or in our case through regularization) discourages instances where a single feature is represented by multiple neurons (Elhage et al., 2022). (In contrast, expanding model capacity has been observed in to encourage so-called *polysemantic* neurons to split into several *monosemantic* neurons in LLMs.)

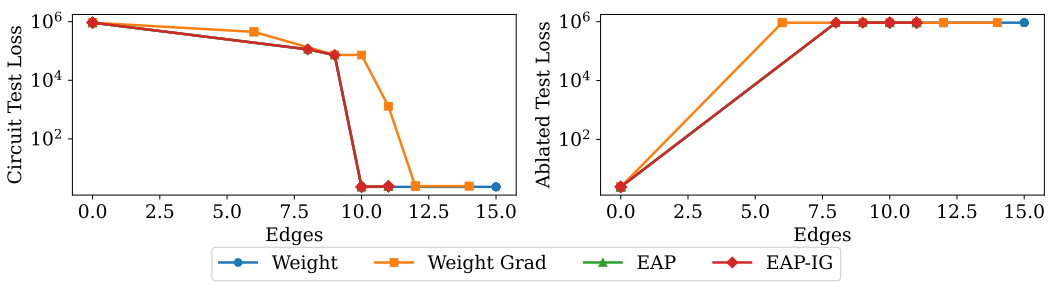


Figure 11: Circuit loss (left) and ablated circuit loss (right) for circuits found using Weight, WEIGHT-GRAD, EAP, and EAP-IG ($m = 20$) with $K = 1, 2, \ldots, 10$ for Bellman-Ford networks. From top to bottom, seeds $0, \ldots, 4$

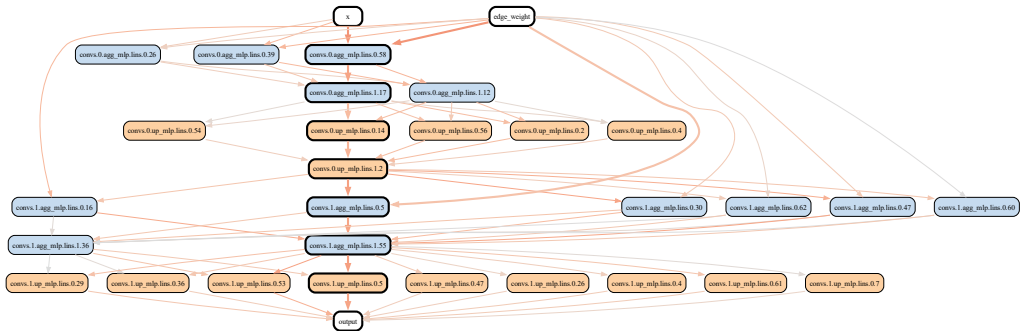


Figure 12: Bellman-Ford circuit at epoch 1000. The circuit contains 73 edges and achieves $\mathcal{L}_{\text{Mult}} = 0.0578$. Elements which are part of the final circuit are bolded.

Pruning vs. Grokking Our findings have potential implications for how the *grokking* phenomenon (Power et al., 2022) may occur in NAR settings like (Nerem et al., 2026) and (Wittig et al., 2026). In general, several mechanisms have been put forth to explain why grokking occurs, including regularization (Power et al., 2022; Nanda et al., 2023; Zhong et al., 2023; Li et al., 2025), representation learning (Liu et al., 2022; 2023; Zhong et al., 2023; Mallinar et al., 2025), information-theoretic phenomena (Clauw et al., 2024; DeMoss et al., 2025), or other properties of training dynamics (Barak et al., 2022; Kumar et al., 2024).

Our pruning observations are most similar to the relationship between model capacity and neural representations observed by Elhage et al. (2022): as model size grows, *polysemantic* neurons (which are responsible for mutiple concepts) split into *monosemantic* neurons (which are responsible for only one concept). Expanding model capacity further allows for redundant representations, where a single concept may be represented by multiple neurons. Here, we observe that the extra parameters early in training are a necessary part of the neural circuit, implying that the representation for a specific quantity (the estimate at an intermediate step of Bellman-Ford, for example) is split among multiple neurons. That is, rather than forming and pruning early heuristic circuits like in (Li et al., 2025), we observe that regularization serves to refine a generalizing model into a more concise circuit of *monosemantic* neurons.

This delayed pruning also has implications for techniques like model compression (Buciluundefined et al., 2006; Han et al., 2016) or distillation (Hinton et al., 2015; Salimans & Ho, 2022; Panigrahi et al., 2025), which seek to emulate the behavior of a large performant model using a much smaller model. The smaller model may be pruned directly from the larger model, or trained using outputs from the larger model in a student-teacher manner. Our results show that strong test performance—even out-of-distribution performance—may not be enough to characterize a desirable teacher model. Rather, we show that continued training with L_1 regularization past model generalization can produce a model with sparser, more efficient circuits, in turn producing a model more amenable to compression.

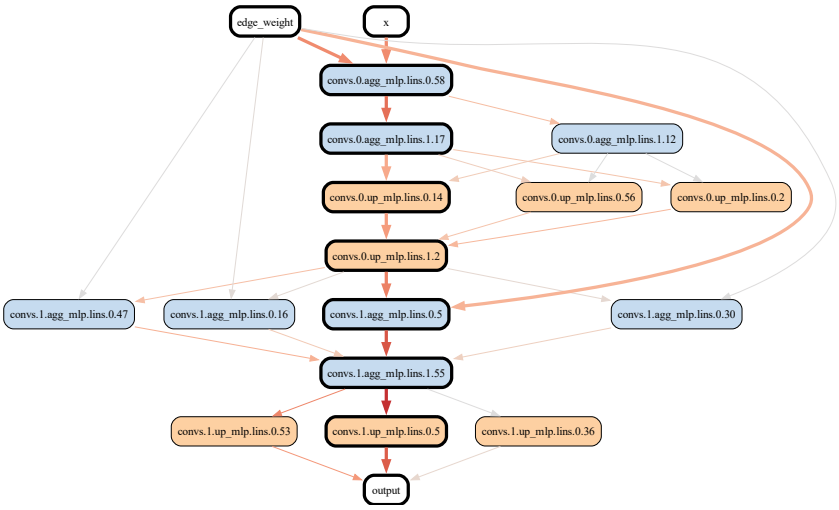


Figure 13: Bellman-Ford circuit at epoch 2000. The circuit has 29 edges and achieves $\mathcal{L}_{\text{Mult}} = 0.0565$. Elements which are part of the final circuit are bolded.

B.4 Bellman-Ford without Self-Loops

Our early experiments in Section 5 did not provide self-loops on training and testing graphs. As a result, while the minimum aggregation in Equation (8) returned the optimal neighbor embedding for each node, it did not include the node's own embedding. Hence the update MLP f_{Up} became responsible for comparing between a node's current shortest path distance and that of its best neighbor. While self-loops are present in Section 5, we briefly study the behavior of the model trained without self-loops to highlight how MINAR reveals the emergence of this ability to compute the minimum between two elements.

The MinAggGNN is trained on the same data as Section 5, achieving an MSE Loss of $\mathcal{L}_{\text{MSE}} = 0.0002$ and $\mathcal{L}_{\text{Mult}} = 0.0642$. The circuit (Figure 15), consisting of 18 edges and discovered using EAP-IG ($m = 20$) achieves a test loss of $\mathcal{L}_{\text{Mult}} = 0.0643$.

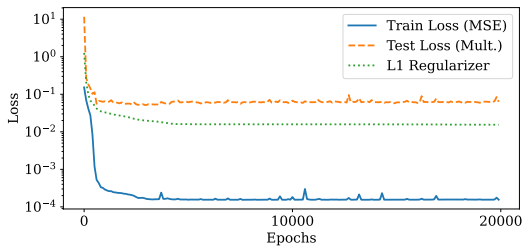


Figure 14: Train loss, test loss, and L_1 regularization term for Bellman-Ford MinAggGNN trained without self-loops.

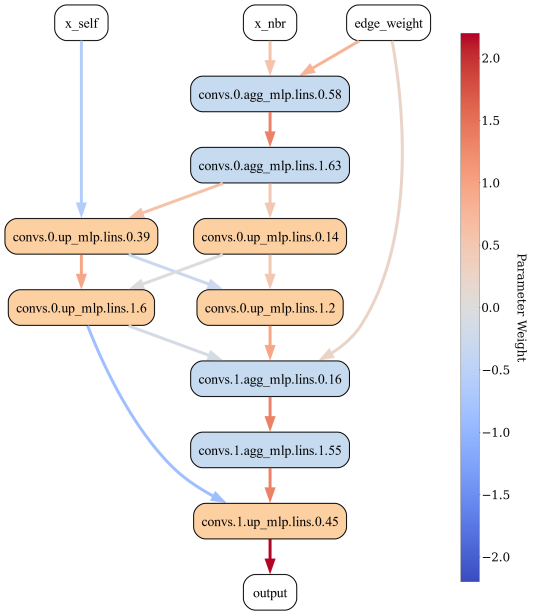


Figure 15: Bellman-Ford circuit from MinAggGNN trained without self-loops. Contains 17 of 18,240 edges. $\mathcal{L}_{\text{Mult}} = 0.0599$.

Here, we wish to highlight the nodes for the first update MLP $f_{\text{Up}}^{(0)}$, (beginning with `convs.0.up_mlp`). Manually inspecting the weights, we see that this portion of the circuit implements the minimum operation between the initial node embedding x_{self} and the best possible update among its neighbors in the first step, which we will denote x_{nbr}^*.

$$\texttt{convs.0.up_mlp.lins.1.6} = \begin{bmatrix} 0 & 0.9620 \end{bmatrix} \text{ReLU}\left(\begin{bmatrix} 0.4921 & 0 \\ 0.5905 & -0.6102 \end{bmatrix} \begin{bmatrix} x_{\text{nbr}}^* \\ x_{\text{self}} \end{bmatrix} \right) \tag{26}$$

If $0.5905x_{\text{nbr}}^* - 0.6102x_{\text{self}} > 0$ (i.e. if roughly $x_{\text{nbr}}^* > x_{\text{self}}$), then this becomes

$$\begin{bmatrix} 0.5681 & -0.5870 \end{bmatrix} \begin{bmatrix} x_{\text{nbr}}^* \\ x_{\text{self}} \end{bmatrix} \propto x_{\text{nbr}}^* - x_{\text{self}}. \tag{27}$$

Otherwise, if $0.5905x_{\text{nbr}}^* - 0.6102x_{\text{self}} < 0$ (i.e. if roughly $x_{\text{nbr}}^* < x_{\text{self}}$), then this becomes 0.

Notice in Figure 15 the primary input to the second step of Bellman-Ford comes from `convs.0.up_mlp.lins.1.2`. If $x_{\text{nbr}}^* < x_{\text{self}}$, then this is the optimal representation for Bellman-Ford. However, if $x_{\text{nbr}}^* > x_{\text{self}}$, the computation will incorrectly carry forward with x_{nbr}^* instead of x_{self}. In this case, by storing a value proportional to $x_{\text{nbr}}^* - x_{\text{self}}$, the neuron `convs.0.up_mlp.lins.1.6` can be rescaled and subtracted from the input to the final update MLP, serving as a correction factor.

The upshot of this calculation is that the update MLP $f_{\text{Up}}^{(0)}$ essentially learns to implement the minimum operation by

$$\min\{x_{\text{nbr}}^*, x_{\text{self}}\} = x_{\text{nbr}}^* - \text{ReLU}(x_{\text{nbr}}^* - x_{\text{self}}). \tag{28}$$

We also highlight the generalization behavior of the circuit in Figure 16. While the full model achieves strong generalization performance early, as in Section 5, the final circuit does not generalize until much later (around epoch 18000). Examining a circuit extracted from an earlier checkpoint (epoch 10000) in Figure 17, we can see that the connections which are pruned away at this later inflection point (namely, the connections to and from `convs.0.up_mlp.lins.0.2`) are partially responsible for this same minimum operation in the first update MLP. As in Appendix B.3, we see L_1 regularization encourage the model to align its internal representations with the neuron basis to achieve a sparser solution.

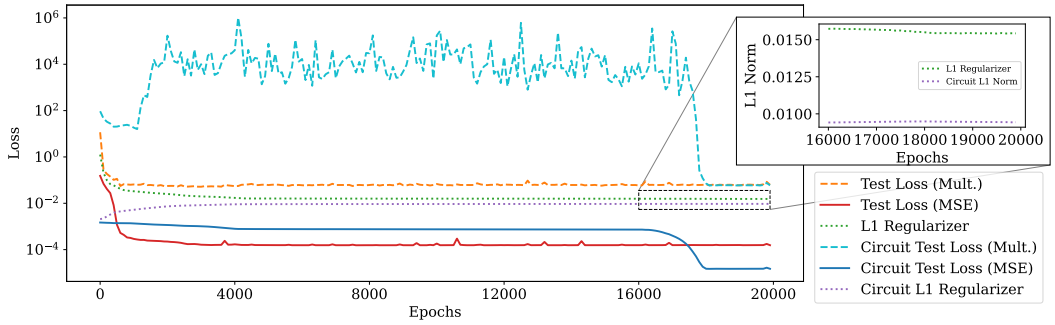


Figure 16: Multiplicative test loss, MSE test loss, and L_1 regularization terms for the full Bellman-Ford network without self-loops and the Bellman-Ford circuit without self-loops. Inset shows a small inflection point in L_1 norms.

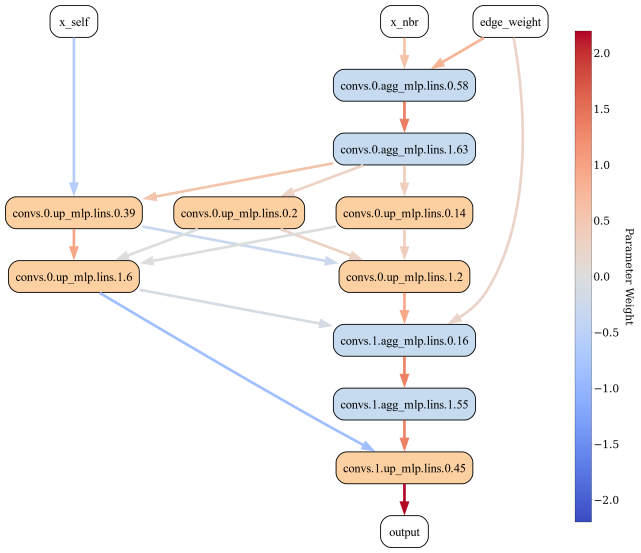


Figure 17: Bellman-Ford circuit extracted from epoch 10000 of MinAggGNN training without self-loops. Contains 22 of 18,240 edges.

B.5 Results from Different Runs

Here we show relevant plots from different experimental trials.

B.5.1 Bellman-Ford

We show circuits extracted across different initializations for experiments in Section 5.2 in Section B.5.1. Each circuit is functionally identical to the mainline run, but contain different neurons. Each circuit is also both sufficient and necessary, as seen in Table 2. Section B.5.1 shows the sizes of circuits found using different methods, showing WeightGrad requires slightly more edges than other scorers in each case.

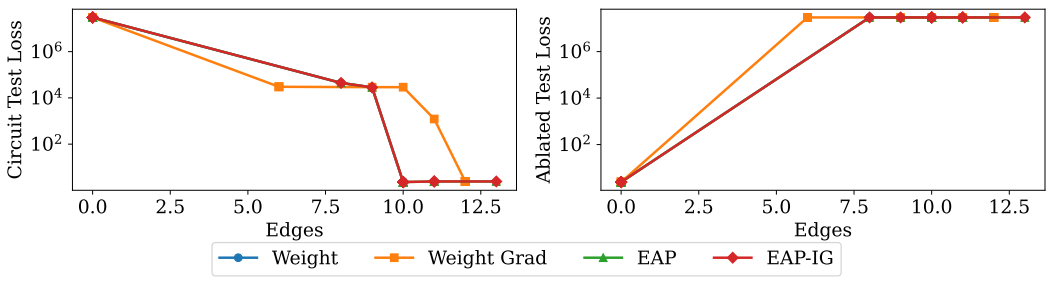


(a) Seed 1. (b) Seed 2. (c) Seed 3. (d) Seed 4.

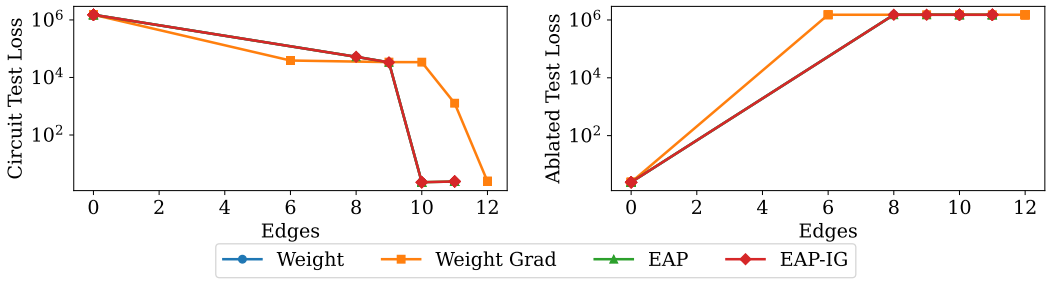
Figure 18: Bellman-Ford circuits across four additional initializations.

Table 2: Bellman-Ford model $\mathcal{L}_{\text{Mult}}$, circuit $\mathcal{L}_{\text{Mult}}$, and ablated $\mathcal{L}_{\text{Mult}}$ for additional initializations.

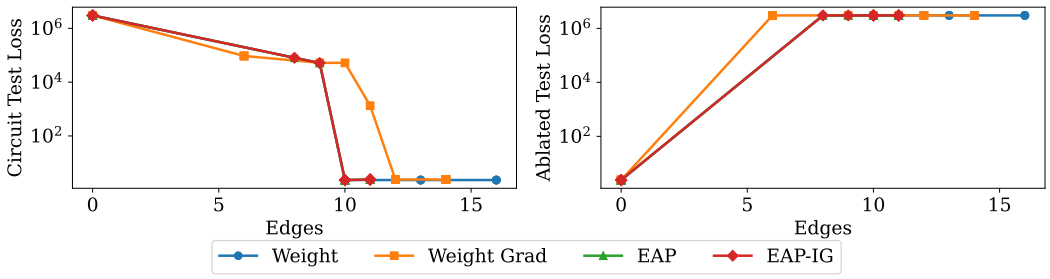
Seed	Model $\mathcal{L}_{\text{Mult}}$	Circuit $\mathcal{L}_{\text{Mult}}$	Ablated $\mathcal{L}_{\text{Mult}}$
1	0.0577	0.0543	698918.0000
2	0.0579	0.0540	35846.6523
3	0.0569	0.0543	71579.8516
4	0.0589	0.0542	21196.0332



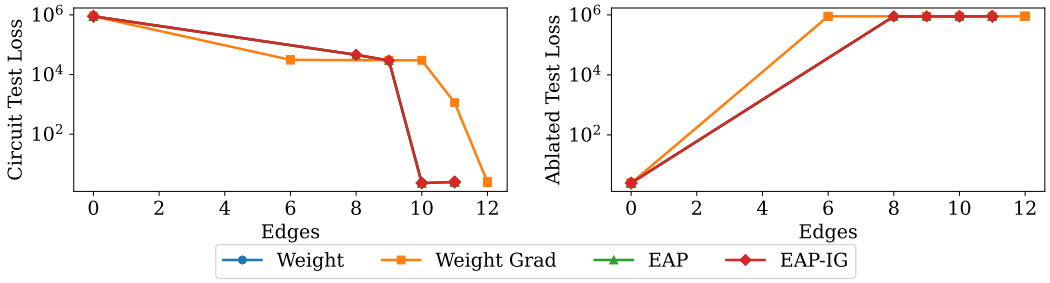
(a) Seed 1



(b) Seed 2

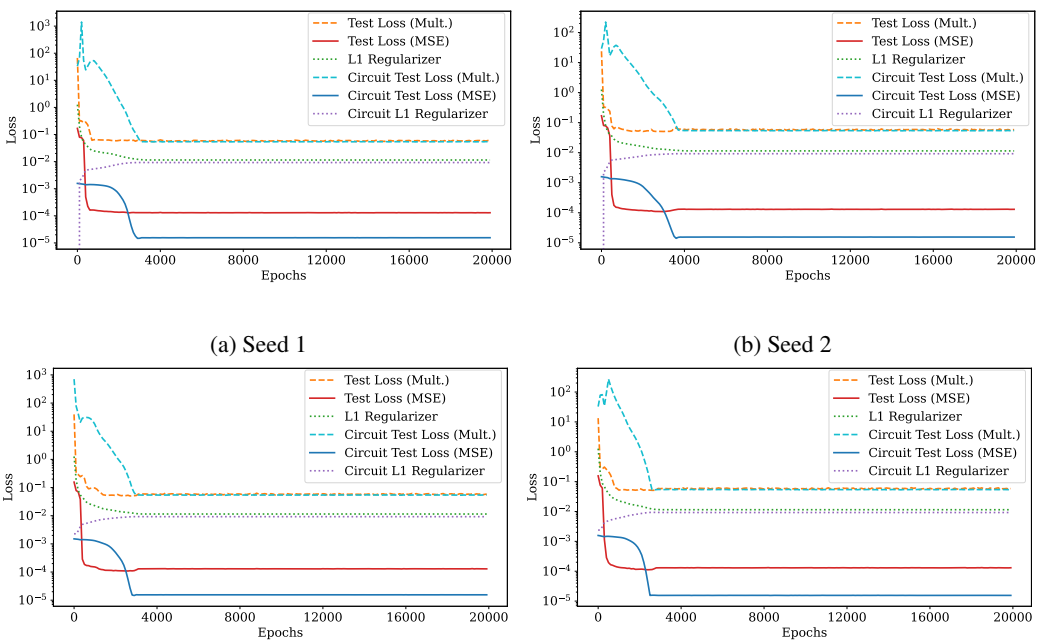


(c) Seed 3



(d) Seed 4

Figure 19: Bellman-Ford circuit sizes found by different scoring methods from different initializations.



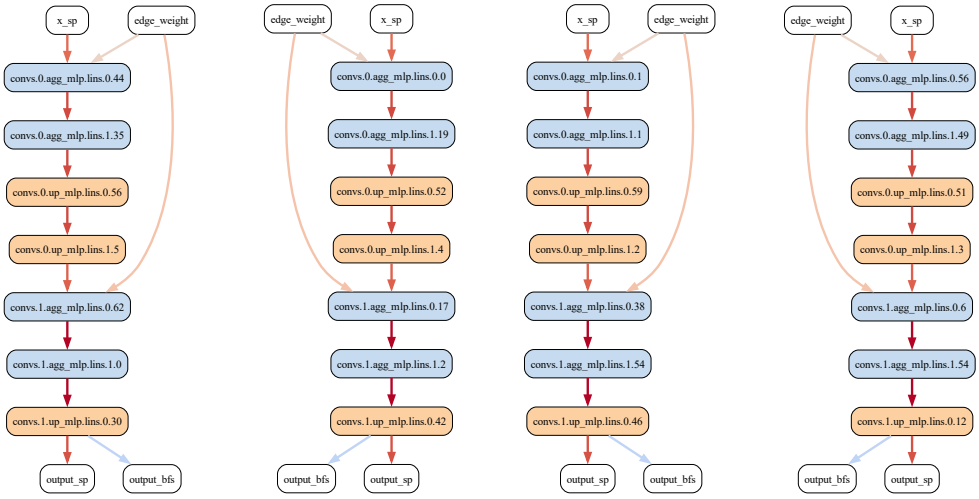
(a) Seed 1 (b) Seed 2

(c) Seed 3 (d) Seed 4

Figure 20: Bellman-Ford circuit generalization behavior across four additional initializations.

B.5.2 Bellman-Ford and Breadth-First Search

We show circuits extracted across different initializations of the Bellman-Ford and BFS network (Section B.5.2), as well as their performance compared to the full model (Table 3). Again, each circuit is functionally the same as the mainline run, but uses different neurons.



(a) Seed 1 (b) Seed 2 (c) Seed 3 (d) Seed 4

Figure 21: Parallel Bellman-Ford and BFS circuits across four additional initializations.

Table 3: Parallel Bellman-Ford and BFS model, circuit, and circuit ablated performance from different initializations.

	Bellman-Ford $\mathcal{L}_{\text{Mult}}$			BFS Accuracy (%)		
Seed	Model	Circuit	Ablated	Model	Circuit	Ablated
1	0.0604	0.0533	36308.5703	100.00	98.32	81.61
2	0.0519	0.0507	8855.6826	100.00	98.31	81.61
3	0.0499	0.0508	18173.6738	100.00	98.31	81.61
4	0.0602	0.0542	25983.5723	100.00	98.32	81.61

B.6 Parameter Summaries

Here we plot parameter summaries for trained Bellman-Ford models from Section 5 and Appendix B.4. For readability we only plot results from the mainline run.

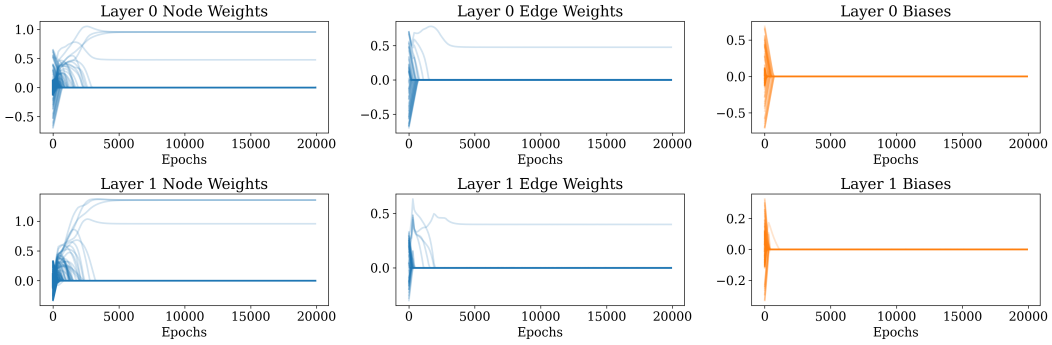


Figure 22: Parameter summary for Bellman-Ford MinAggGNN (Section 5.2).

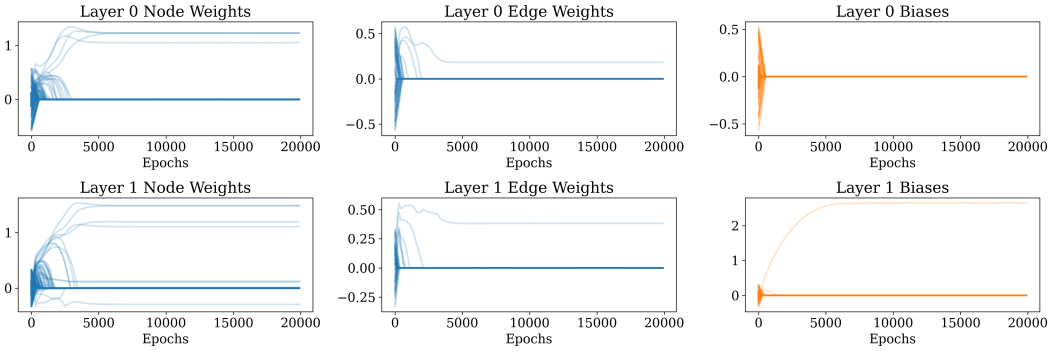


Figure 23: Parameter summary for parallel Bellman-Ford and BFS MinAggGNN (Section 5.2).

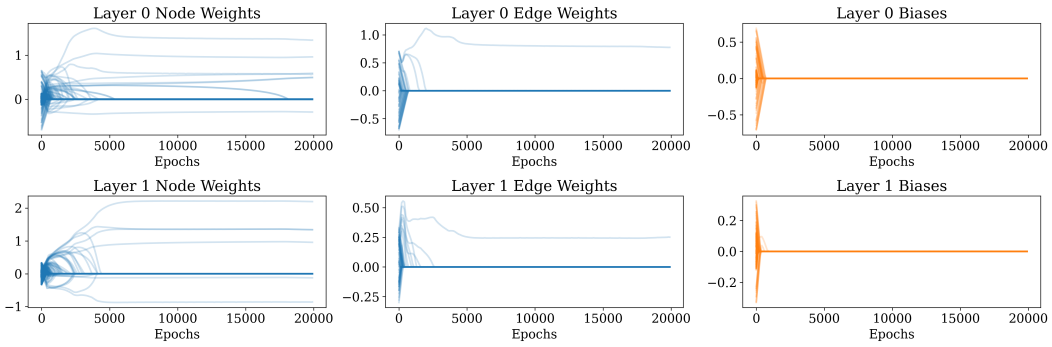


Figure 24: Parameter summary for Bellman Ford MinAggGNN without self-loops (Appendix B.4).

Table 4: Mean ± standard deviation validation and out-of-distribution test performance of maximum-aggregated GINE on breadth-first search (BFS), depth-first search (DFS), Dijkstra's algorithm, Prim's minimum spanning tree (MST), Bellman-Ford (B-F), Articulation Points (Art. Points), and Bridges across three seeds.

	Node Accuracy (%)					Node F_1	Edge F_1
Algorithm	BFS	DFS	Dijkstra	Prim's MST	B-F	Art. Points	Bridges
Validation	99.84 ± 0.04	75.39 ± 2.83	89.97 ± 1.08	82.17 ± 1.34	89.70 ± 1.17	69.37 ± 4.28	66.21 ± 2.11
Test	96.51 ± 0.66	22.73 ± 1.19	66.84 ± 1.72	47.93 ± 2.47	64.05 ± 1.06	31.49 ± 9.47	22.82 ± 3.21

C Additional Results for SALSA-CLRS

Here we provide additional results from Section 6, including model training (Figure 26) curves, validation and test performance (Table 4), and sparsity (Figure 25) of the SALSA-CLRS GNN averaged over 3 seeds. We also report circuit overlap for the main run at different sparsity (Figure 28), and separate fidelity metrics (Figure 27; Figure 30; Figure 32) as well as characterization scores on other seeds (Figure 29; Figure 31) for each attribution patching method at different circuit sizes in each run. Figure 33 shows the results of the BFS circuit formation experiment for the additional seeded runs. Figure 34 and Figure 35 show the circuit overlaps for additional seeds.

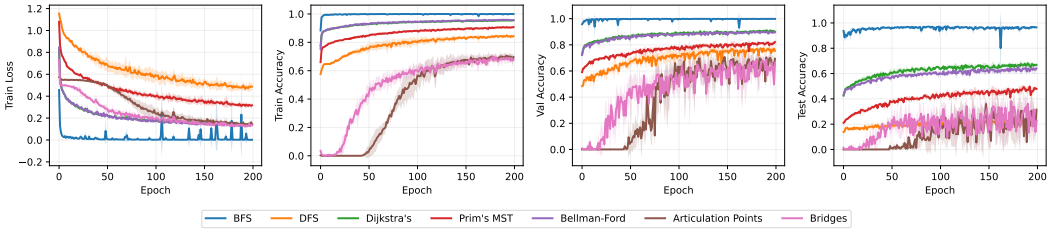


Figure 25: From left to right: train loss, train accuracy, validation accuracy, OOD test accuracy of SALSA-CLRS models.

We report the positive and negative fidelity here, which are combined into the *characterization score* in Figure 3 from Amara et al. (2022), although we adapt the definitions slightly to measure circuit fidelity rather than fidelity of input graphs. The *positive fidelity* Fid^+ measures the *necessity* of a circuit C while the *negative fidelity* Fid^- measures the *sufficiency*: denote by $\Phi^{\setminus C}$ the model without the circuit and by Φ^C the model using only the circuit weights:

$$\begin{aligned}\text{Fid}^+(C) &= 1 - \frac{1}{|V|}\sum_{v\in G}\mathbf{1}\{\Phi_v(G) = \Phi_v^{\setminus C}(G)\}\\ \text{Fid}^-(C) &= 1 - \frac{1}{|V|}\sum_{v\in G}\mathbf{1}\{\Phi_v(G) = \Phi_v^{C}(G)\}.\end{aligned} \tag{29}$$

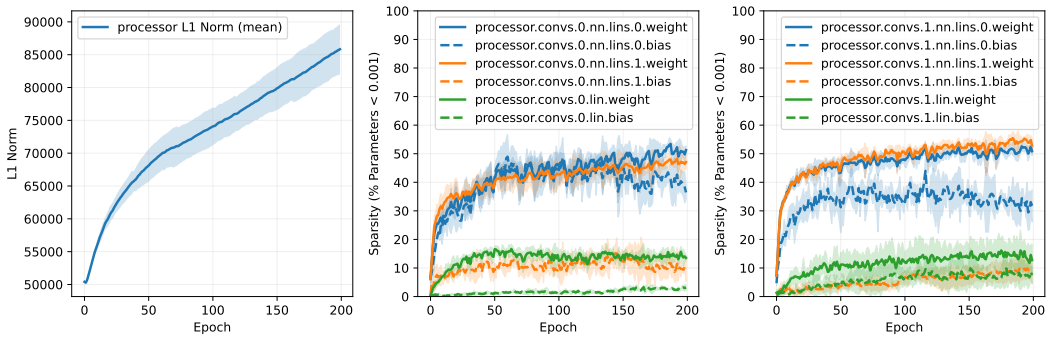


Figure 26: L_1 norm (left), layer 1 parameter sparsity (middle), and layer 2 parameter sparsity (right) of SALSA-CLRS GNNs.

Here, Fid^- measures how often the circuit disagrees with the full model. We report $(1 - \text{Fid}^-)$, which is also frequently referred to as *faithfulness* in the circuit discovery literature. Fid^+ measures how often ablating the circuit changes the model prediction.

We combine these metrics using the *characterization score* from Amara et al. (2022). The characterization score is the harmonic mean of $\text{Fid}^+(C)$ and $(1 - \text{Fid}^-(C))$:

$$\textsc{Char}(C) = \left[\frac{1}{2\text{Fid}^+(C)} + \frac{1}{2(1 - \text{Fid}^-(C))}\right]^{-1}. \tag{30}$$

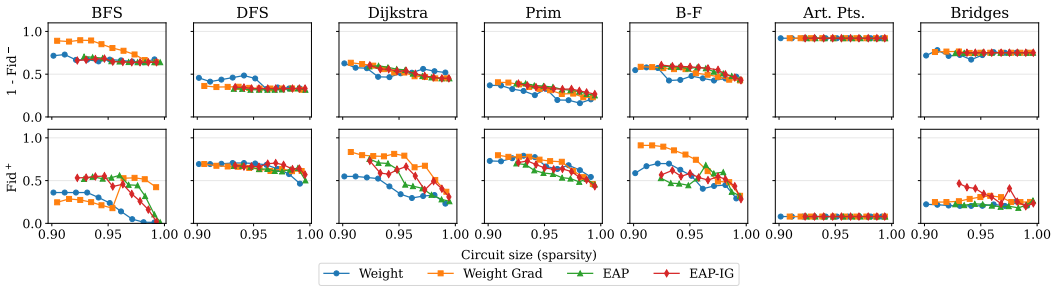


Figure 27: Sparsity vs. $(1 - \text{Fid}^-)$ (top) and Fid^+ (bottom) for SALSA-CLRS circuits (Seed 0) using Weight, WEIGHTGRAD, EAP, and EAP-IG ($m = 20$). Higher is better.

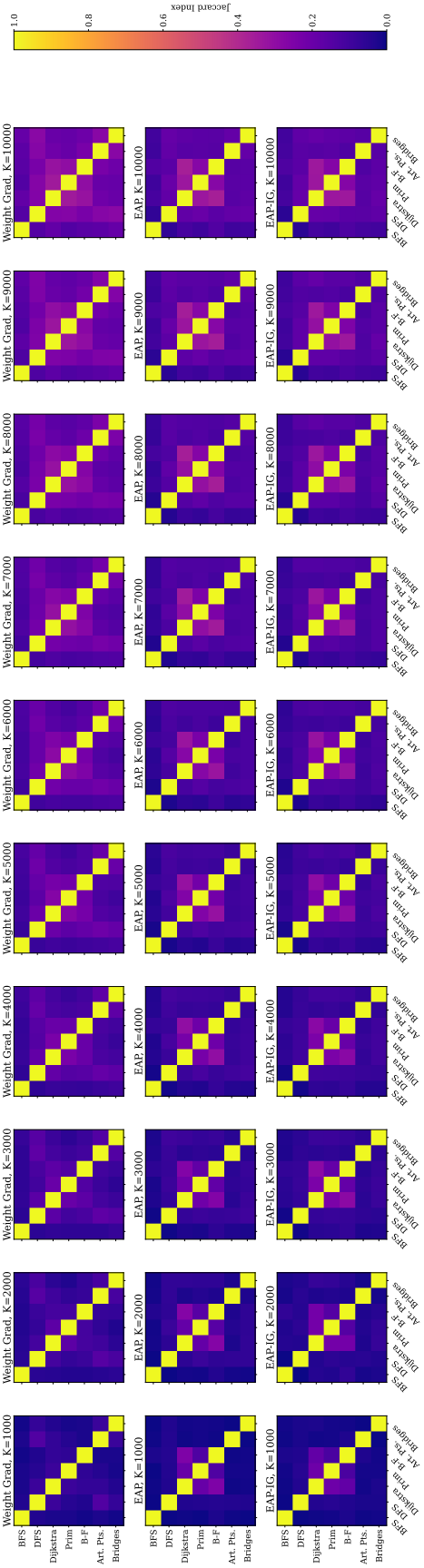


Figure 28: Weighted Jaccard index between mainline run CLRS circuits using WEIGHTGRAD, EAP, and EAP-IG ($m = 20$) at $K \in \{1000, 2000, \ldots, 10,000\}$.

C.1 Fidelity for Additional Initializations

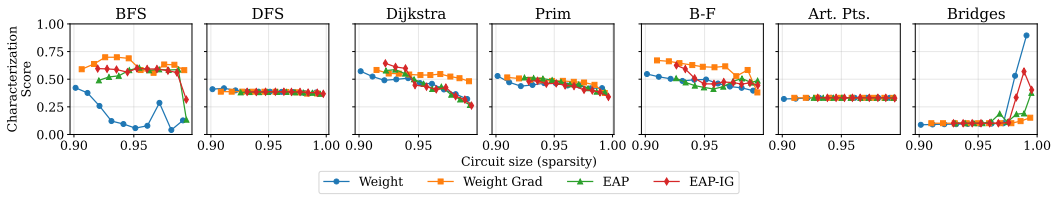


Figure 29: Sparsity vs. Characterization Score for SALSA-CLRS circuits (Seed 1).

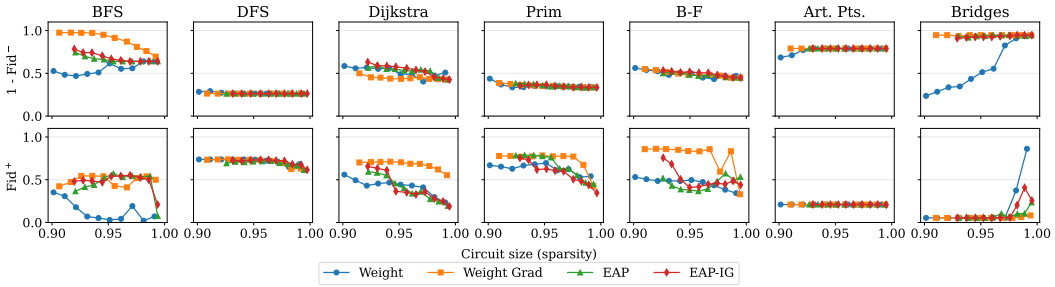


Figure 30: Sparsity vs. $(1 - \text{Fid}^-)$ (top) and Fid^+ (bottom) for SALSA-CLRS circuits (Seed 1).

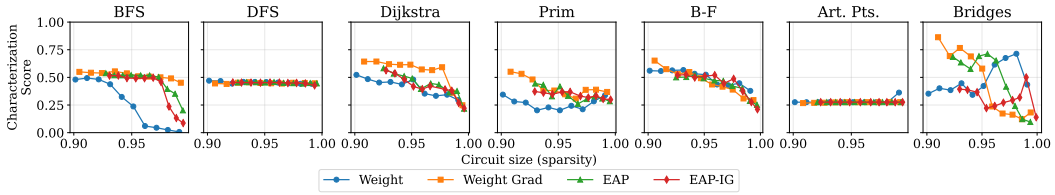


Figure 31: Sparsity vs. Characterization Score for SALSA-CLRS circuits (Seed 2).

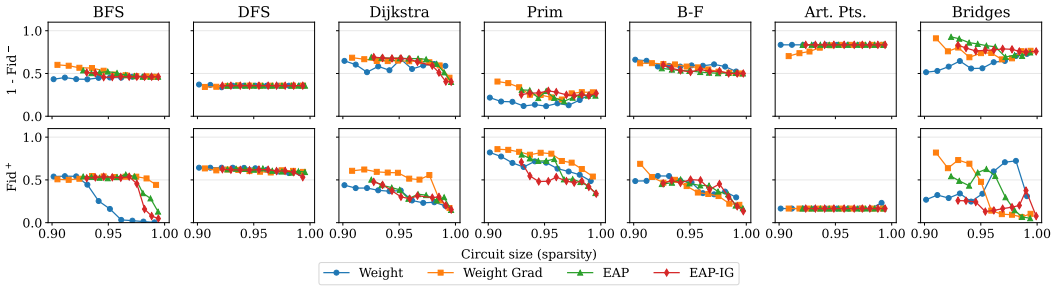
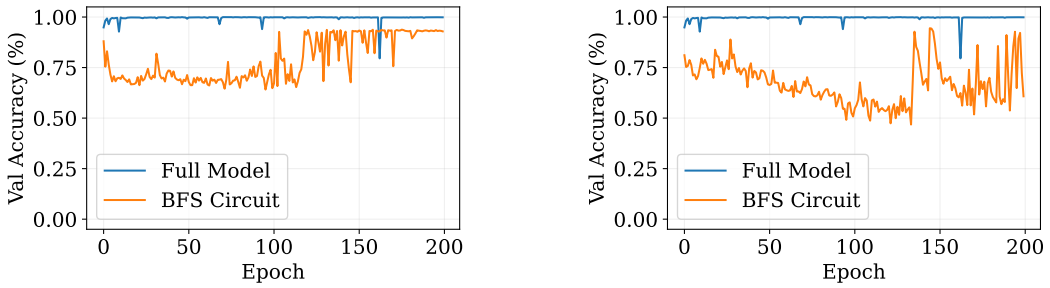


Figure 32: Sparsity vs. $(1 - \text{Fid}^-)$ (top) and Fid^+ (bottom) for SALSA-CLRS circuits (Seed 2).



(a) BFS circuit performance for Seed 1

(b) BFS circuit performance for Seed 2

Figure 33: BFS Circuit performance on additional seeded runs. Seed 1 (left) contains similar phase changes to Figure 4, while Seed 2 (right) never reaches the final phase.

D Training Data

D.1 Shortest Path Data

Following Nerem et al. (2026), we construct a training set $\mathcal{G}_{\text{train}}$ from the graphs depicted in Figure 36 for $K = 2$. It includes all path graphs of the form $P^{(1)}_{K+1}(a, \ldots, b, \ldots, 0)$ for $a, b \in \{0, 1, \ldots, 2K\} \times \{0, \ldots, 2K + 1\}$ (where b is the weight of the K-th edge). It also includes the graph $H^{(0)}_K$ from Figure 36 and the special path graphs $P^{(0)}_1(1)$, $P^{(1)}_2(1, 0)$. We also include extra path graphs: four three-node path graphs initialized at step zero of Bellman-Ford and four four-node path graphs initialized at step two of Bellman-Ford, each with randomly generated edge weights.

For the out-of-distribution test set, we include a collection of 3 and 4-node cycle graphs; complete graphs on 5 to 200 nodes; and Erdős-Rényi graphs on 5 to 200 nodes with $p = 0.5$. To provide extra examples of graphs with unreachable nodes, we also generate binary and ternary trees of depths of 3 and 4. All test graphs have randomly generated edge weights, and the test set contains 300 graphs in total. The corrupted dataset includes versions of every graph in the test dataset with zero-ablated features. An example is given in Figure 37.

D.2 SALSA-CLRS

We use the implementation of Minder et al. (2023) to generate data for BFS, DFS, Dijkstra's algorithm, and Prim's algorithm. We also generate data for Bellman-Ford, Articulation Points, and Bridges from Veličković et al. (2022). (We use implementations of these tasks contained in the codebase for (Minder et al., 2023), but are not in the paper.)

Each algorithmic task has a training set consisting of 1000 graphs of $n \in \{4, 7, 11, 13, 16\}$ nodes. This includes 250 Erdös-Renyi graphs of with $p \sim \text{Unif}(\log n/n, 3 \log n/n)$, 250 Barabási-Albert graphs with $m \in \{1, 2, 3\}$, 250 regular trees of arity $r \in \{2, 3, 4\}$, and 250 barbell graphs whose connecting paths are of length $l \in \{1, 2, 3\}$. The 1000 validation graphs are drawn from the same distributions with $n = 16$ nodes, and we test out-of-distribution generalization by using a test set of 1000 Erdös-Renyi graphs on $n = 80$ nodes with the same ranges for the edge probability p. For circuit discovery experiments, we use an additional 128 unseen graphs drawn from the validation distribution.

Following Minder et al. (2023), we use only graph tasks where hints are bound to the given graph structure. For example, in BFS, the hints for each node are pointers to its predecessor in the BFS tree, which must correspond to edges in the actual graph. In contrast, some tasks have nodes maintain pointers to arbitrary nodes. These pointers may not be to immediate neighbors of each node, and hence the pointers may not correspond to actual edges in the graph.

We describe below the exact inputs and outputs for each task, as well as the ablation and loss we use for circuit discovery. In addition to the losses below, we have an additional loss term of $\lambda_{\text{hidden}} = 0.1$

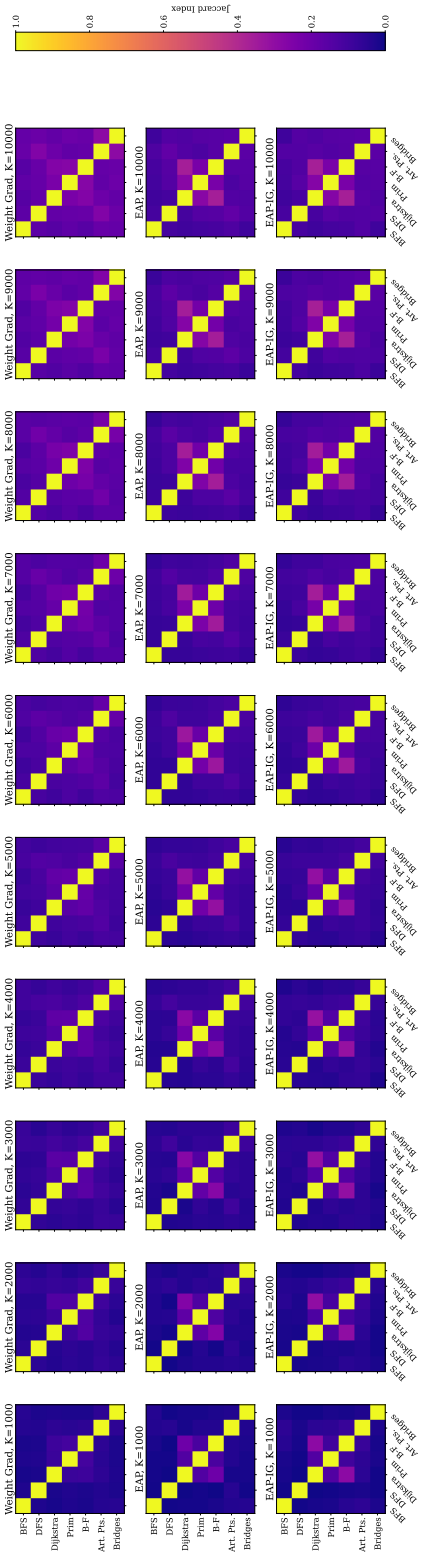


Figure 34: Weighted Jaccard index between Seed 1 CLRS circuits.

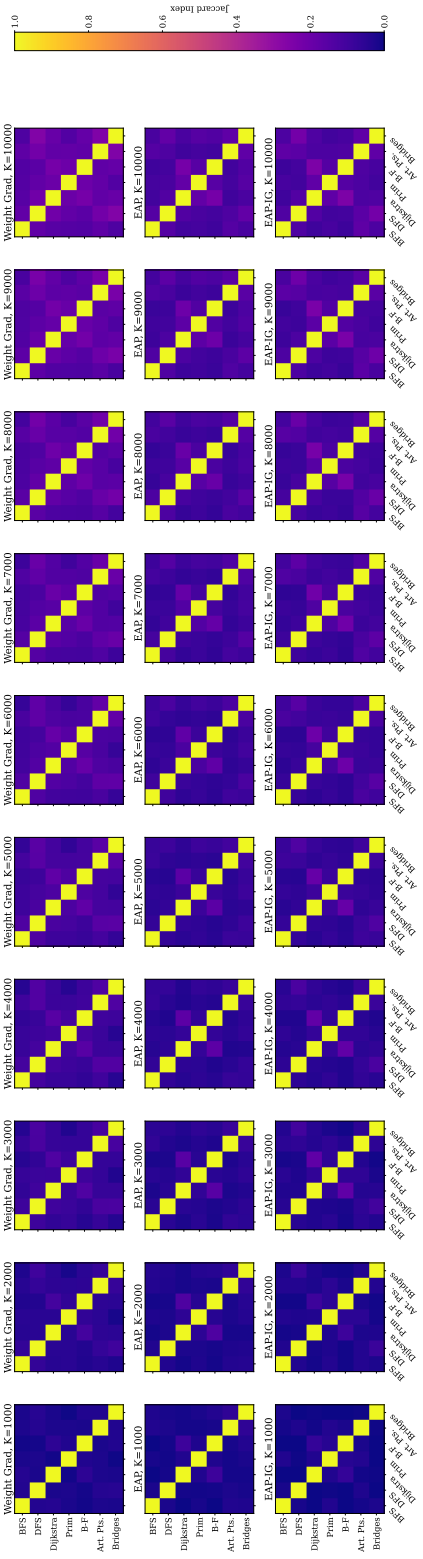


Figure 35: Weighted Jaccard index between Seed 2 CLRS circuits.

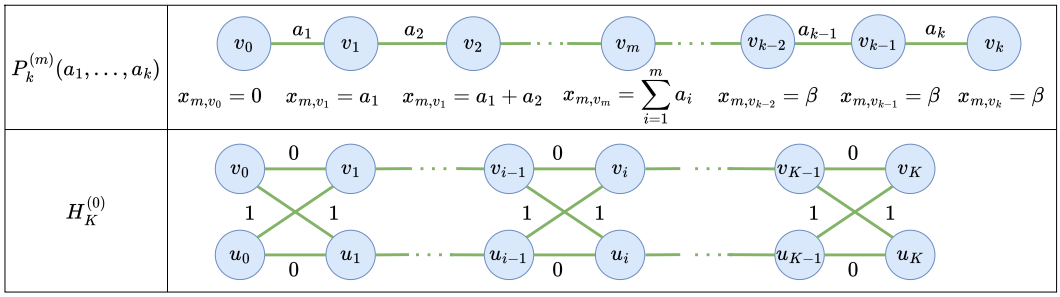


Figure 36: Training graphs for the Bellman-Ford MinAggGNN from Nerem et al. (2026).

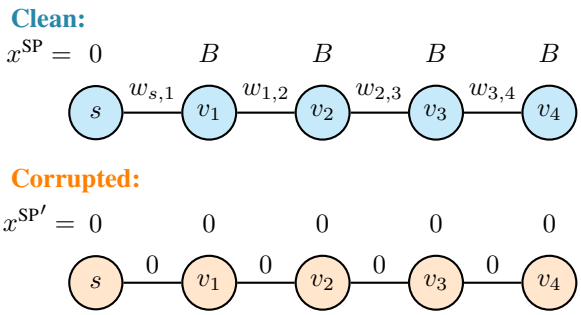


Figure 37: Example of clean and corrupted data for shortest path length prediction on a path graph.

times the MSE between hidden node embeddings on the clean and corrupted data. Note also that every task contains node positions, assigned evenly in [0,1).

Breadth-First Search (Moore, 1959). The input consists of a flag for the source node. The output for each node is the edge to its predecessor in the directed BFS tree. The corrupted data removes the source flag and node positions. The loss is cross-entropy loss across predicted probabilities for each edge.

Depth-First Search (Moore, 1959). The input consists of a flag for the source node. The output for each node is the edge to its predecessor in the directed DFS tree. The corrupted data removes the source flag and node positions. The loss is cross-entropy loss across predicted probabilities for each edge.

Dijkstra's Shortest Path (Dijkstra, 1959) The input consists of a flag for the source node and edge weights on the graph. The output for each node is the edge to its predecessor in the shortest path from the source. The corrupted data removes the source flag, node positions, and edge weights. The loss is cross-entropy loss across predicted probabilities for each edge.

Prim's MST (Prim, 1957) The input consists of a flag for the source node and edge weights on the graph. The output for each node is the edge to its predecessor in the minimum spanning tree. The corrupted data removes the source flag, node positions, and edge weights. The loss is given by cross-entropy loss across predicted probabilities for each edge.

Bellman-Ford Shortest Path (Bellman, 1958) The input consists of a flag for the source node and edge weights on the graph. The output for each node is the edge to its predecessor in the shortest path from the source. The corrupted data removes the source flag, node positions, and edge weights. The loss is given by cross-entropy loss across predicted probabilities for each edge.

Articulation Points (Cormen et al., 2022) The input consists only of the graph structure and node positions. The task is to classify whether or not each node is an articulation point, i.e. whether its removal would disconnect the graph. The corrupted data removes the node positions. The loss is given by binary cross-entropy loss across predicted probabilities for each node.

Bridges (Cormen et al., 2022) The input consists only of the graph structure and node positions. The task is to classify whether or not each edge is a bridge, i.e. whether its removal would disconnect the

graph. The corrupted data removes the node positions. The loss is given by binary cross-entropy loss across predicted probabilities for each edge.

D.3 Designing Corrupted Datasets

One major component of edge attribution patching is the design of probing datasets of clean and corrupted inputs. Each corrupted input must be able to align with its corresponding clean input to facilitate computation edge scoring. (This alignment is either between sequences in the LLM case or the graph structure in our case.) At the same time, the corruption must specifically target the behavior of interest. In the LLM setting, Dai et al. (2022) distill factual information recall into ⟨Head, Relation, Tail⟩ triples. Wang et al. (2023) isolate indirect object identification by swapping a single name or pronoun in a sentence.

In general, for algorithmic tasks the relevant information is given to the network in the form of node features and edge weights. Thus, we mostly follow this intuition of changing tokens without changing structure by ablating features without changing the structure of our corrupted graphs. This way, on each clean-corrupted pair, the graph neural network performs the same message-passing behavior for each node while the behavior of interest (processing based on node features like node positions or edge weights) is changed. While we use zero ablation for our experiments, we note that there are many valid ablations. For example, when using replacing features with random values (uniformly drawn between 0 and 1) instead of zero-ablating, we find that there is no difference in the resulting circuit performance (Figure 38), even though the actual attribution patching values differ. This is because, broadly, circuit selection algorithms like Algorithm 1 depend only on the ranking of circuit components rather than exact score values.

However, we also note that the space of possible data corruptions is even larger for graph data. For example, one could consider perturbing the graph structure as well as the graph features, since the result will still be a valid input to the GNN. (This of course differs from text: a graph of the same nodes and no edges is a valid corruption in the GNN setting whereas a sentence consisting entirely of empty tokens ceases to be a sentence.) This is a particularly interesting possibility for CLRS tasks like articulation points or bridges, which are purely structural and do not depend on any features like edge weights or source designations.

E Model Training and Hardware

Computations for Section 5 and Appendix B.4 were performed on an NVIDIA RTX A6000 Laptop GPU using PyTorch Paszke et al. (2019) and PyTorch-Geometric Fey & Lenssen (2019). For each MinAggGNN, $f_{\text{Agg}}^{(\ell)}$ and $f_{\text{Up}}^{(\ell)}$ are implemented as two-layer MLPs with a hidden dimension of 64. The intermediate dimension of each network is 8. We train each model using AdamW Loshchilov & Hutter (2019) with a learning rate of $\gamma = 0.001$ for 20000 epochs using full batches, a default weight decay of $\lambda = 0.01$, and an L_1 regularization term of $\eta = 0.001$.

Because the parallel Bellman-Ford and BFS experiment uses the same training and testing sets as the Bellman-Ford experiment, the positive and negative reachability classes are imbalanced. (About 91.24% of training nodes and 81.98% of test nodes are reachable.) Therefore, we use class weighting so that the positive and negative classes have equal weight during training. We additionally scale the BCE loss by a factor of 25 during training, as we observe that the MSE and L_1 terms dominate training otherwise.

The SALSA-CLRS experiments in Section 6 were performed partially on an HPC cluster using 7 NVIDIA A100 GPUs, and partially on the same NVIDIA RTX A6000 Laptop GPU. Model training took place entirely on the HPC cluster: each task is trained on a single GPU and gradients are synced using PyTorch's distributed data parallel module. The model was trained for 200 epochs using a minibatch size of 32 with AdamW and a learning rate of $\gamma_0 = 0.001$. To encourage sparsity, we use a weight decay of $\lambda = 0.1$ and an L_1 regularization term of $\eta = 0.0001$. Circuits were identified using both the cluster and the laptop GPU, but most evaluation and analysis was performed on the laptop.

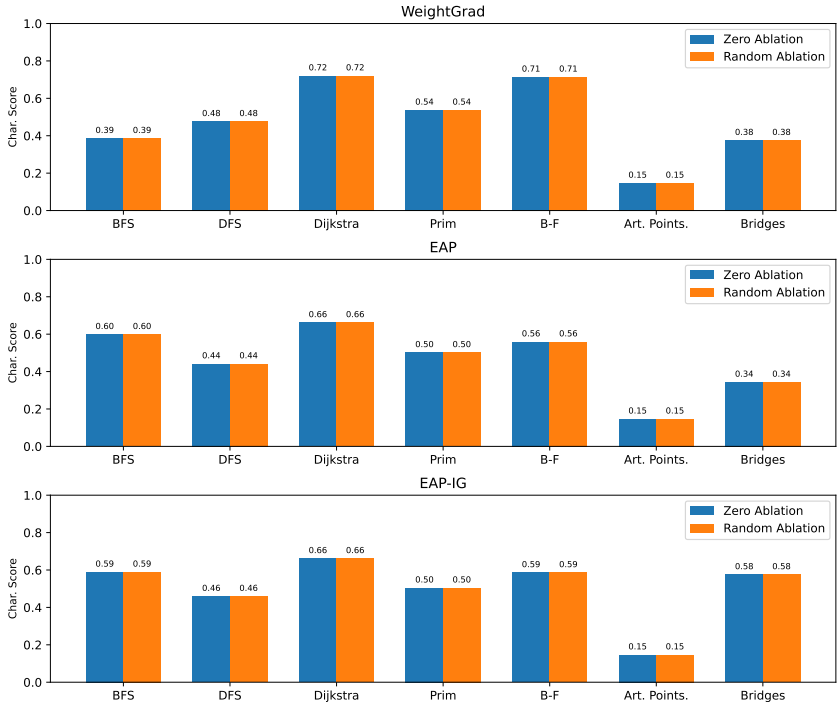


Figure 38: Characterization scores of circuits found at $K = 10000$ using zero ablation and random ablation. The resulting circuits are nearly identical.