MINAR: Mechanistic Interpretability for Neural Algorithmic Reasoning

Jesse He[1,2] **Helen Jenne**[1] **Max Vargas**[1] **Davis Brown**[1,3]
Gal Mishne[2] **Yusu Wang**[2] **Henry Kvinge**[1,4]

[1] Pacific Northwest National Laboratory, [2] University of California San Diego
[3] University of Pennsylvania, [4] University of Washington
`jeh020@ucsd.edu`

Abstract

Graph neural networks (GNNs) are known to be capable of implementing specific algorithmic steps that guarantee strong out-of-distribution performance, a property referred to as *algorithmic alignment* or *neural algorithmic reasoning* (NAR). At the same time, recent advances in the reasoning capabilities of large language models (LLMs) have created an interest in *mechanistic interpretability*: identifying specific model components that are responsible for certain tasks. In this work, we adapt circuit discovery methods from mechanistic interpretability to the GNN setting with **M**echanistic **I**nterpretability for **N**eural **A**lgorithmic **R**easoning (MINAR). We validate MINAR by applying it to two GNNs: one predicting single-source shortest path distances and another computing shortest path distances and reachability in parallel. Through both examples, we demonstrate how mechanistic interpretability can offer fine-grained insight into an algorithmically aligned model. Our code will be made available at `https://github.com/he-jesse/minar`.

1 Introduction

The recent surge in the capabilities of large language models (LLMs) has created a commensurate demand for novel interpretability methods suited to analyze these models. At the same time, the discovery of phenomena like *grokking* [13] has shown that surprising structure arises in trained models. This confluence of interests has led to the emergence of *mechanistic interpretability* [9]: attempting to reverse-engineer a model's behavior in terms of its internal processes. At the same time, researchers have noticed that graph neural networks (GNNs) are capable of implementing specific algorithms. This is thought to be a result of the resemblance between their message-passing structure and dynamic programming. This notion, dubbed "algorithmic alignment" [17] or "neural algorithmic reasoning" [16], promises robust out-of-distribution generalization and substantial improvements in sample complexity. With an interest in discovering how models perform specific algorithmic tasks on the one hand, and a class of models that implement specific algorithms on the other, the algorithmic alignment of graph neural networks 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), a method to identify neuron-level circuits in GNNs using circuit discovery methods like EAP [15] and EAP-IG [5]. To our knowledge, MINAR is the first attempt to bring mechanistic interpretability to the GNN setting. We apply our method to two GNNs: one predicting single-source shortest path distances and another computing shortest path distances and reachability in parallel. In the first, we validate our method by identifying the circuit that implements the Bellman-Ford algorithm. In the second, we identify that the model learns Bellman-Ford, but recycles much of the shortest path computation to predict reachability. These examples demonstrate how neuron-level circuits can offer detailed insight to trained models.

2 Related Work

Circuit Discovery and Mechanistic Interpretability Early mechanistic interpretability work (e.g., [9, 21]) 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. To this end, recent work has focused on methods to partially or totally automate much of the mechanistic interpretability process, detecting specific model components that are responsible for certain tasks. Such components and their connections are dubbed a "circuit" by Olah et al. [11], and the process of identifying circuits is known as *circuit discovery*. A number of approaches for circuit discovery have been proposed, broadly performing some form of activation patching [15, 5, 20] or pruning [1, 2, 19]. These methods formulate the model as a *computation graph* and attempt to find small subgraphs that are 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 or detecting which of two numbers is greater, and identifies coarse-grained circuits whose nodes are entire attention heads or MLP modules.

Algorithmic Alignment in Graph Neural Networks While mechanistic interpretability has revealed surprising structure in large generalist models, a particular class of narrow models has also been shown to possess algorithmic "reasoning" capabilities: graph neural networks. Broadly, GNNs operate on graph data via an iterative *message-passing* scheme: each node in the graph has an embedding, and at each layer of the network, each node will aggregate the embeddings of its neighbors and use the aggregated embeddings to update its own embedding. Xu et al. [17] and Dudzik and Veličković [3] point out that this aggregate-and-update flow resembles dynamic programming, hypothesizing that the *algorithmic alignment* of graph neural networks affords them an advantage in learning algorithmic tasks like single-source shortest path computation. Further, Veličković et al. [16] show that learning BFS and Bellman-Ford in parallel can help GNNs generalize on both tasks. However, it was only recently that Nerem et al. [10] showed that GNNs are not only *capable* of learning specific algorithms, but that a properly designed GNN will *provably* learn a specific algorithm (in this case Bellman-Ford) during training.

3 Preliminaries

It will be necessary to disambiguate the term "computation graph," which researchers define differently in the circuit discovery and in the graph neural network literature. Let Φ be a GNN operating on a 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. Where necessary, we will denote by Ψ an arbitrary (perhaps non-graph) neural network.

In the circuit discovery literature [2, 15, 20], a "computation graph" (which we will call a *model computation graph*) refers to a graph representing individual model components. These model components are typically defined at an architectural level, rather than at the mathematical level. While earlier circuit discovery work defines these components at a lower resolution (e.g., entire attention heads or feedforward MLP modules), we consider circuits at the level of individual neurons.

Definition 3.1 (Model Computation Graph)**.** Let Ψ be a neural network with arbitrary subcomponents ψ_i. The *model computation graph* of Ψ, denoted $G_C^m(\Psi)$, is the directed graph with vertices i and edges (i, j) if the output of ψ_i is part of the input to ψ_j.

In graph neural network literature, some authors [18, 6] use "computation graph" to represent the layered message-passing performed by a GNN. We first 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 $v \in V(G)$ and $\ell = 0, \dots, L$, with layer 0 being the input node features $\Phi_v^{(0)} = X_v$. Each subsequent layer is given by

$$\Phi_v^{(\ell+1)} = f_{\mathrm{Up}}^{(\ell)}\left(\Phi_v^{(\ell)}, \bigoplus_{(u,v)\in E(G)} f_{\mathrm{Agg}}^{(\ell)}(\Phi_u^{(\ell)})\right) \tag{1}$$

where $f_{\mathrm{Agg}}^{(\ell)}$ and $f_{\mathrm{Up}}^{(\ell)}$ are any function 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.

Definition 3.2 (Message-Passing Computation Graph)**.** Let Φ be an L-layer GNN operating on a graph G. Then the *(message-passing) computation graph* $G_c^v(\Phi)$ for Φ at $v \in G$ is the directed graph with nodes $\bigsqcup_{\ell=0}^{L} \mathcal{N}^{(\ell)}(v)$ and edges $(i^{(\ell-1)}, j^{(\ell)})$ for each $(i,j) \in E(G)$ if $j \in \mathcal{N}^{(L-\ell)}(v)$ where $\mathcal{N}^{(\ell)}(v)$ denotes the ℓ-hop neighborhood of v.

In what follows, we will follow the circuit discovery literature and use "computation graph" to refer to the model computation graph. Furthermore, we will consider circuit discovery at the neuron level: each node in the computation graph corresponds to an individual neuron in the model, and each edge corresponds to a connection between neuron activations.

4 Circuit Discovery for GNNs

Having introduced our definitions and notation, we introduce the scoring methods EAP [15] and EAP-IG [5], both forms of *attribution patching*. The goal of attribution patching is to approximate the difference in prediction that would occur if an edge in the computation graph was corrupted or removed [5]. Actually removing the edge and obtaining the result with a full forward pass is referred to as *activation patching* [2]. Our choice to use attribution patching rather than activation patching is in large part due to efficiency, as our neuron-level computation graphs are typically very dense. We also describe in this section our approach to discovering neuron-level circuits in the GNN setting.

4.1 Attribution Patching Scores

We will use the model computation graph using a single node prediction as the output, with the aggregated neighbor information appearing as additional inputs. We adopt the method of edge attribution patching (EAP) [15] and its integrated gradients variation EAP-IG [5]. Formally, 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. The attribution score of (i,j) for a prediction $\Psi(x)$ takes the activation z_i at ψ_i for the input x together with the activation z_i' from a corrupted input x'. Then, given a loss L which measures the distance between predictions $y = \Psi(x)$ and $y' = \Psi(x')$, the EAP score uses the gradient of L with respect to the output of ψ_j. That is,

$$\mathrm{EAP}_{(i,j)}(x,x') = (z_i' - z_i)^\intercal \frac{\partial}{\partial \psi_j} L(\Psi(x), \Psi(x')). \tag{2}$$

Equation (2) comes from the first-order term in a Taylor expansion for the perturbation performed by activation patching. 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 backwards pass for each pair of inputs [15].

EAP-IG [5] adapts EAP to use the integrated gradients method of [14], approximating an integral over the straight-line path between z_i and z_i' with a Riemann sum of m terms:

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

EAP-IG is m times slower than EAP, essentially performing the EAP calculation m times for each pair of inputs [5].

4.2 Circuit Identification in GNNs

Because a graph neural network operates on each node of the input graph, EAP and EAP-IG assign each computation edge a different score for each input node in the graph. To give each computation edge a single score for a given prediction, MINAR sums over the scores from each vertex to obtain the total score for each computational edge on one graph instance. That is,

$$\mathrm{EAP}_{(i,j)}(G, G') = \sum_{v \in G} \left({z'}_i^{v'} - z_i^v \right)^\intercal \frac{\partial}{\partial \psi_j} L(\Phi_v(G), \Phi_{v'}(G')) \tag{4}$$

where z_i^v is the activation of neuron ψ_i on node v. We compute EAP-IG in a simlar manner, again summing over nodes to give a total score on a single graph instance. For EAP-IG, we implement the steps between the original and corrupted inputs by interpolating between all node and edge features.

Finally, since attribution scores are typically averaged over multiple input instances to understand their behavior across a probing dataset, we average over each input graph to obtain the final attribution score for each edge. We also follow previous work in using the absolute value of each attribution score for downstream circuit discovery:

$$\mathrm{EAP}_{(i,j)} = \left| \frac{1}{|\mathcal{G}_{\text{probe}}|} \sum_{(G,G') \in \mathcal{G}_{\text{probe}}} \mathrm{EAP}_{(i,j)}(G, G') \right| \tag{5}$$

and similarly for EAP-IG. We note that the summation in (4) can be replaced by any pooling operation (e.g. mean). We choose summation to mirror the test loss (9) in Section 5.1 is computed, following Nerem et al. [10].

To guarantee connectedness of the identified circuit, we propose to construct the circuit from complete paths between the computation graph's inputs and outputs. While previous work in circuit discovery uses naive top-k selection [15] or a greedy Dijkstra-like algorithm [5], these methods often produce parentless or childless edges which must then be pruned. Instead, MINAR takes advantage of the fact that the computation graph is directed, acyclic, and topologically sorted, and therefore supports an efficient longest-path algorithm. Each longest path computation takes $O(|V(G_c^m)| + |E(G_c^m)|)$ time, the same runtime as constructing the model computation graph G_c^m itself.

Using the absolute value of our attribution scores as weights, we initialize our circuit with the longest path from any input to each model output. Then, continuing down the top-k edges which have not yet been included, we compute the longest path containing that edge and add it to the circuit. Thus once scores are computed, finding a circuit that includes the top k edges takes time $O((k+d^L)(|V(G_c^m)|+|E(G_c^m)|))$, where d^L is the output dimension. This way, MINAR guarantees that the identified circuit is a connected subgraph of the computation graph whose only parentless and childless nodes are the model inputs and outputs, respectively.

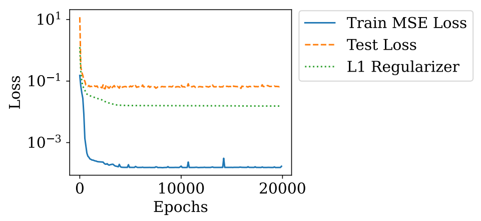
5 Results

We demonstrate our circuit discovery method using two classical algorithmic tasks in the neural algorithmic reasoning literature: single-source shortest paths (via Bellman-Ford) and node reachability (via breadth-first search). We test first shortest paths by itself, then test shortest paths and reachability in parallel, as in [16]. In the shortest path task, we validate our method by identifying the circuit predicted by Nerem et al. [10] to implement the Bellman-Ford algorithm. For the parallel task, we examine the hypothesis of Veličković et al. [16] that a model predicting shortest path distance and reachability in parallel may experience some amount of transfer between tasks, reusing part of the computation for one task to perform the other.

Both tasks share the same training set, constructed as in [10], but with additional node features and labels to support the reachability task. Specifically, each node is given two initial features based on the typical initialization for Bellman-Ford and breadth-first-search:

$$x_v^{\text{SP}} = \begin{cases} 0 & v \text{ is the source} \\ B & \text{otherwise.} \end{cases} \quad \text{and} \quad x_v^{\text{BFS}} = \begin{cases} 1 & v \text{ is the source} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Here B is any large number greater than any path length, which we take to be $B = 1000$. For the shortest path GNN, only x_v^{SP} is provided as input to the model. When the two tasks are performed in parallel, the initial node feature is 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 discuss



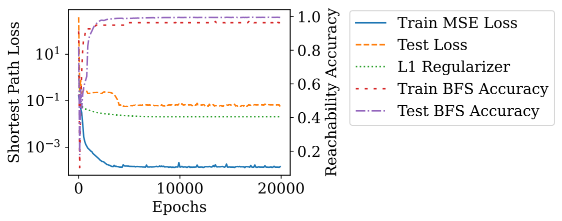


Figure 1: MSE training loss, multiplicative test loss, and L_1 regularization term for Bellman-Ford MinAggGNN (left) and parallel Bellman-Ford and BFS MinAggGNN (right). The right plot also shows train and test accuracy on the BFS reachability task.

the dataset in greater detail in Appendix B. In our circuit discovery experiments, we corrupt the input data by setting every edge weight to zero and flipping the input features:

$$x_v^{\mathrm{SP}\prime} = \begin{cases} B & v \text{ is the source} \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad x_v^{\mathrm{BFS}\prime} = \begin{cases} 0 & v \text{ is the source} \\ 1 & \text{otherwise.} \end{cases} \tag{7}$$

We also perturb the edge weights, setting them to zero in the corrupted data. In both experiments, we evaluate the identified circuits by performing inference using just the circuit components. We achieve this by zeroing out network parameters that are not included in the identified circuit and performing a forward pass with the ablated network.

5.1 Bellman-Ford

We validate MINAR by replicating [10]. We train a minimum-aggregated message-passing network (which [10] dubs a MinAggGNN) to predict single-source shortest path distances. We use a two-layer network, corresponding to two steps of Bellman-Ford. Each layer is given by

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

where we implement $f_{\mathrm{Agg}}^{(\ell)}$ and $f_{\mathrm{Up}}^{(\ell)}$ as two-layer MLPs and we use the minimum operation as the aggregator to mimic the Bellman-Ford algorithm. By training with L_1 sparsity regularization and a small curated training set, [10] shows that such a GNN must implement the Bellman-Ford algorithm. While we use MSE during training, we follow [10] in evaluating using a percentage-based test loss:

$$L_{\text{test}} = \frac{1}{|\mathcal{G}_{\text{test}}|} \sum_{G\in\mathcal{G}_{\text{test}}} \sum_{v\in G \text{ reachable}} \left|1 - \frac{y_v}{\Phi(v)}\right| \tag{9}$$

where N is the number of reachable nodes from the source of each graph and y_v is the true label for each reachable node. The trained GNN reaches an MSE of $L_{\mathrm{MSE}} = 0.0002$ and a test loss of $L_{\text{test}} = 0.0673$. We plot the loss and the L_1 regularization term in Figure 1, showing that the model has converged to a sparse, generalizable solution.

Having trained a model to implement a shortest-path solution, we use EAP-IG with $m = 20$ steps to identify the responsible circuit (Figure 2). The resulting circuit consists of only 17 edges out of 18240 edges in the full computation graph. For comparison, a minimum of 10 parameters are predicted by [10]. The circuit, taken as a subnetwork of the full model, achieves a loss of 0.0644 on the test set, matching the high performance of the full model. In fact, the circuit achieves a slightly lower loss overall, although not substantially. In this example, MINAR successfully recovers a small circuit that captures the behavior of the full model, validating our approach.

5.2 Bellman-Ford and breadth-first search in parallel

We expand the Bellman-Ford example by introducing a second task: predicting reachability by breadth-first search. As with Bellman-Ford, the model's two layers correspond to two steps of BFS. The combination of Bellman-Ford with BFS was investigated by Veličković et al. [16], who note that both algorithms traverse a graph in the same manner, and hence hypothesize that learning one task can benefit performance on the other.

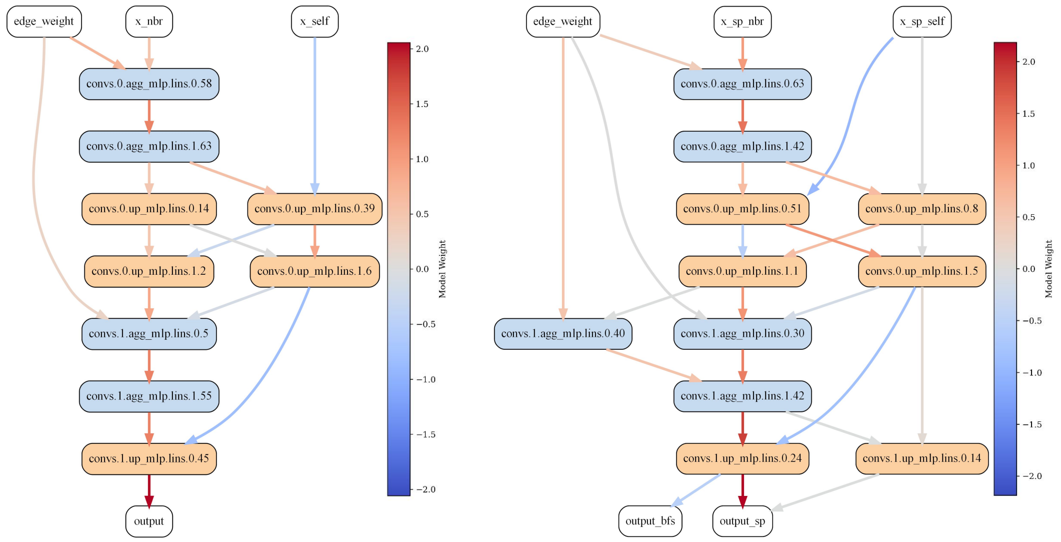


Figure 2: Identified circuits in the Bellman-Ford MinAggGNN (left) and parallel Bellman-Ford with BFS MinAggGNN (right). Nodes are individual MinAggGNN neurons. Input and output neurons are colored white, f_{Agg} neurons are colored blue, and f_{Up} neurons are colored orange. Circuit edges are colored by corresponding model weights.

We train simultaneously, using MSE for the shortest path task and binary cross-entropy for the reachability task. Because we use 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 observed that the MSE and L_1 terms dominated training otherwise. Figure 1 shows the loss and accuracy curves of the model during training, with a test loss of 0.0604 on shortest path and a test accuracy of 0.9943 on reachability. We again use EAP-IG with $m = 20$ steps to identify the circuit (Figure 2). We identify a circuit with 25 edges (out of 18432) which achieves a test loss of 0.0635 on shortest paths and a reachability accuracy of 0.9664.

Examining the circuit for the parallel GNN in Figure 2, we notice the nodes `x_sp_nbr` and `x_sp_self` correspond to the shortest path input feature. The BFS feature is absent, indicating that the model does not actually use the BFS feature x^{BFS}. Instead, the model simply recycles steps from the shortest path computation: comparing the two circuits in Figure 2 reveals that the Bellman-Ford computation follows a very similar structure in both models. However, in the final layers of the parallel model, the same neuron `convs.1.up_mlp.lins.0.24` is responsible for the bulk of both outputs. Thus we see that rather than fully implement BFS and Bellman-Ford in parallel, the parallel GNN uses a heuristic from the shortest path calculation to perform the reachability prediction.

6 Conclusion

We propose MINAR, bringing methods from mechanistic interpretability to the setting of neural algorithmic reasoning with graph neural networks. To our knowledge, our work is the first to attempt circuit discovery in GNNs to study NAR. Through two case studies, we demonstrate how MINAR can be used to identify circuits implementing Bellman-Ford and BFS in trained GNNs, two classical tasks in NAR. We chose these settings because the learned algorithms are well-understood and clearly interpretable, allowing us to validate our method. In future work, we intend to apply MINAR to novel algorithmic and mathematical tasks or to algorithmically-aligned tasks that use real data containing high-dimensional features.

Acknowledgments and Disclosure of Funding

We thank Robert R. Nerem and Samantha Chen for helpful discussions regarding the Bellman-Ford MinAggGNN in [10].

References

[1] 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, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 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/`.

[2] 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, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16318–16352. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf`.

[3] 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, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20635–20647. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/8248b1ded388fcdbbd121bcdfea3068c-Paper-Conference.pdf`.

[4] 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`.

[5] 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`.

[6] 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, page 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`.

[7] Victoria R. Li, Jenny Kaufmann, Martin Wattenberg, David Alvarez-Melis, and Naomi Saphra. Can interpretation predict behavior on unseen data?, 2025. URL `https://arxiv.org/abs/2507.06445`.

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

[9] 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`.

[10] Robert R. Nerem, Samantha Chen, Sanjoy Dasgupta, and Yusu Wang. Graph neural networks extrapolate out-of-distribution for shortest paths, 2025. URL `https://arxiv.org/abs/2503.19173`.

[11] 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. https://distill.pub/2020/circuits/zoom-in.

[12] 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.

[13] 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`.

[14] 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, page 3319–3328. JMLR.org, 2017.

[15] Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery, 2023. URL `https://arxiv.org/abs/2310.10348`.

[16] 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`.

[17] 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`.

[18] 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, pages 10737–10745, 2021. doi: 10.1609/aaai.v35i12.17283.

[19] Lei Yu, Jingcheng Niu, Zining Zhu, and Gerald Penn. Functional faithfulness in the wild: Circuit discovery with differentiable computation graph pruning, 2024. URL `https://arxiv.org/abs/2407.03779`.

[20] 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, 2025. URL `https://arxiv.org/abs/2502.06852`.

[21] 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`.

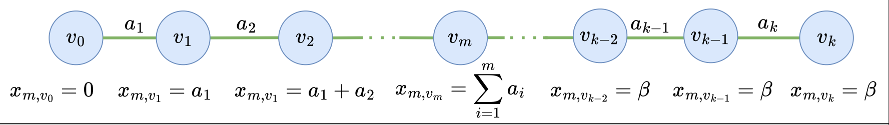
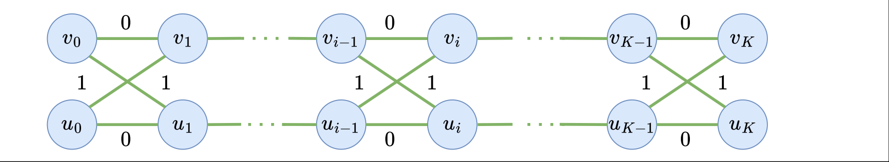
$P_k^{(m)}(a_1, \dots, a_k)$	
$H_K^{(0)}$	

Figure 3: Training graphs for the Bellman-Ford MinAggGNN from [10].

A Model Training

We perform our computations on an NVIDIA RTX A6000 Laptop GPU using PyTorch [12] and PyTorch-Geometric [4]. 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 [8] with a learning rate of $\gamma = 0.001$ for 20,000 epochs using full batches and an L_1 regularization term of $\eta = 0.001$.

B Training Data

Following [10], we construct a training set $\mathcal{G}_{\text{train}}$ from the graphs depicted in Figure 3 for $K = 2$. It includes all path graphs of the form $P_{K+1}^{(1)}(a, \dots, b, \dots, 0)$ for $a, b \in \{0, 1, \dots, 2K\} \times \{0, \dots, 2K + 1\}$ (where b is the weight of the K-th edge). It also includes the graph $H_K^{(0)}$ from Figure 3 and the special path graphs $P_1^{(0)}(1)$, $P_2^{(1)}(1, 0)$. 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 test set, we also largely follow [10]. We include a collection of 3, 4, and 5-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.

C Regularization

Our use of an L_1 sparsity regularization follows Nerem et al. [10], who show that training their MinAgg GNN with L_1 regularization is necessary to induce the desired Bellman-Ford implementation. Regularization is also employed in several mechanistic interpretability works to induce a “correct” implementation in a trained model. For example, Zhong et al. [21] use weight decay when training their model to perform modular addition, and Li et al. [7] show that weight decay is necessary to prune “vestigial” circuits that implement trivial heuristics early in training. The impact of regularization can be seen in the parameter summaries for the trained networks, as shown in Figure 4 and Figure 5.

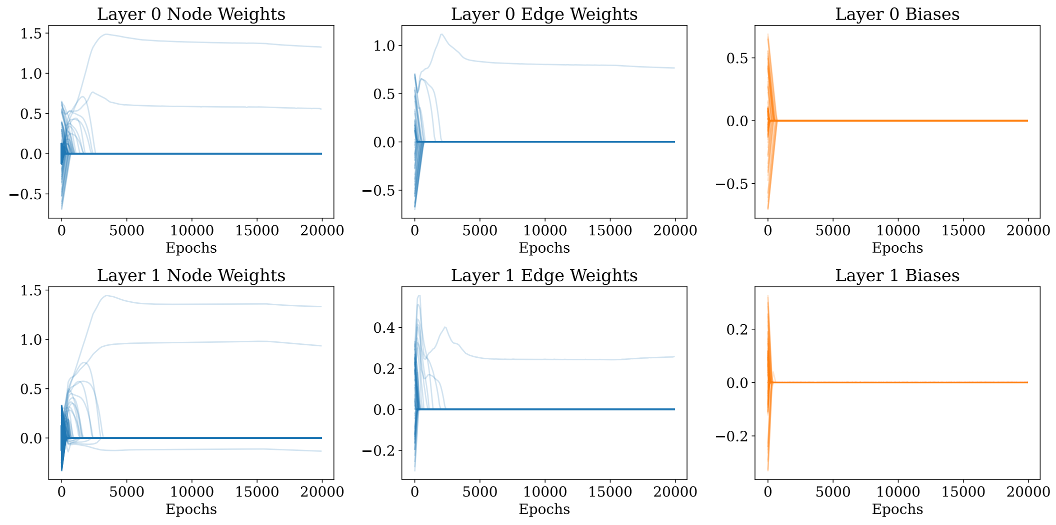


Figure 4: Parameter summaries for each component of the Bellman-Ford MinAggGNN.

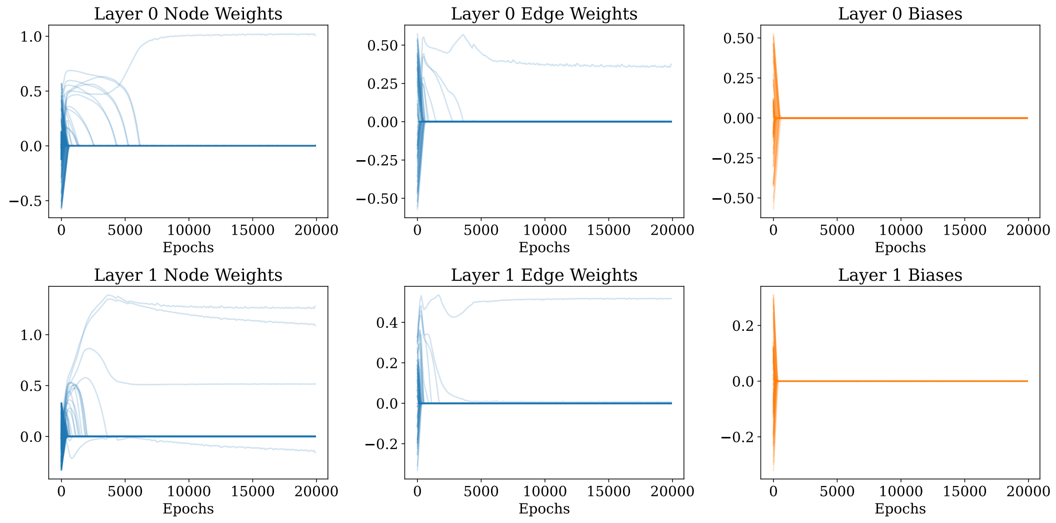


Figure 5: Parameter summaries for each component of the parallel Bellman-Ford and breadth-first search MinAggGNN.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our contributions are described in the abstract and introduction and reflect the content of the rest of the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We address our limitations in the conclusion, and discuss the runtime of our method in Section 4.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This work does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental settings are described in Section 5 and Appendix A. Algorithms are described in Section 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: Due to internal policies, we are unable to release code at this time. However, we intend to provide code by the date of the workshop.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe training details and hyperparameter settings for model training and circuit discovery experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Performing multiple training trials of each experiment would require too much compute. The circuit discovery methods are deterministic, so error bars would not be appropriate.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources used are provided in Appendix A.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We only use synthetic data in our experiments.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Our work is a generic method for circuit discovery in GNNs, and poses no specific societal risks beyond those of machine learning as a whole.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This work is a generic method and poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the relevant code.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the code with documentation.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This work does not involve human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This work does not involve human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: This work does not use an LLM in its core methods.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.