Editable Graph Neural Network for Node Classifications

Zirui Liu*
Rice University
Houston, USA
Zirui.Liu@rice.edu

Zhimeng Jiang*
Texas A&M University
College Station, USA
zhimengj@tamu.edu

Shaochen Zhong
Rice University
Houston, USA
Shaochen.Zhong@rice.edu

Kaixiong Zhou
Rice University
Houston, USA
Kaixiong.Zhou@rice.edu

Li Li
Samsung Electronics America
Mountain View, USA
li.li1@samsung.com

Rui Chen
Samsung Electronics America
Mountain View, USA
rui.chen1@samsung.com

Soo-Hyun Choi
Samsung Electronics America
Mountain View, USA
sh9.choi@samsung.com

Xia Hu
Rice University
Houston, USA
Xia.Hu@rice.edu

ABSTRACT

Graph Neural Networks (GNNs) have achieved prominent success in many graph-based learning problems, such as credit risk assessment in financial networks and fake news detection in social networks. However, the trained GNNs still make errors and these errors may cause serious negative impact on society. *Model editing*, which corrects the model behavior on wrongly predicted target samples while leaving model predictions unchanged on unrelated samples, has garnered significant interest in the fields of computer vision and natural language processing. However, model editing for graph neural networks (GNNs) is rarely explored, despite GNNs' widespread applicability. To fill the gap, we first observe that existing model editing methods significantly deteriorate prediction accuracy (up to 50% accuracy drop) in GNNs while a slight accuracy drop in multi-layer perception (MLP). The rationale behind this observation is that the node aggregation in GNNs will spread the editing effect throughout the whole graph. This propagation pushes the node representation far from its original one. Motivated by this observation, we propose Editable Graph Neural Networks (EGNN), a neighbor propagation-free approach to correct the model prediction on misclassified nodes. Specifically, EGNN simply stitches an MLP to the underlying GNNs, where the weights of GNNs are frozen during model editing. In this way, EGNN disables the propagation during editing while still utilizing the neighbor propagation scheme for node prediction to obtain satisfactory results. Experiments demonstrate that EGNN outperforms existing baselines in terms of effectiveness (correcting wrong predictions with lower accuracy drop), generalizability (correcting wrong predictions for

other similar nodes), and efficiency (low training time and memory) on various graph datasets.

KEYWORDS

Graph neural networks, Editable training

1 INTRODUCTION

Graph Neural Networks (GNNs) have achieved prominent results in learning features and topology of graph data [1–12]. Based on spatial message passing, GNNs learn each node through aggregating representations of its neighbors and the node itself recursively. Once trained, the model is typically deployed as static artifacts to make decisions on a wide range of tasks, such as credit risk assessment in financial networks [13] and fake news detection in social networks [14]. However, the cost of making a wrong decision could be higher in these graph applications. Over-trusted creditworthiness on borrowers can lead to severe loss for lenders, and failure detection of fake news has a serious negative impact on society.

Ideally, it is desirable to correct these serious errors and generalize corrections to similar mistakes, while preserving the model's prediction accuracy on unrelated input samples. To obtain generalization ability for similar samples, the most prevalent method is to fine-tune the model with a new label on the single example to be corrected. However, this approach often spoils the model prediction on other unrelated samples. To cope with the challenge, many model editing frameworks have been proposed to adjust model behaviors by correcting errors as they appear [15–18]. Specifically, these editors usually require an additional training phase to help the model "prepare" for the editing process before applying any edits [15–18].

Although model editing has shown promise to modify vision and language models, to the best of our knowledge, there is no existing work tackling the critical mistakes in graph data. Despite the straightforward concept, it is challenging to efficiently change GNNs' behaviors on the massively connected nodes. First, due to the message-passing mechanism in GNNs, editing the model behavior on a single node can propagate changes across the entire graph, significantly altering the node's original representation, which may destroy the prediction performance on the training dataset. Therefore, compared to the neural networks for computer vision or natural language processing, it is harder to maintain the model prediction on other input samples. Second, unlike other types of neural networks, the input nodes are connected in the graph domain. Thus, when editing the model prediction on a single node using gradient descent, the representation of each node in the whole graph is required [2, 19, 20]. This distinction introduces complexity and computational challenges when making targeted adjustments to GNNs, especially on large graphs.

In this work, we delve into studying the graph model editing problem, which is more challenging than the independent sample edits. We first observe the existing editors significantly harm the overall node classification accuracy although the misclassified nodes are corrected. The test accuracy drop is up to 50%, which prevents GNNs from being practically deployed. We experimentally study the rationale behind this observation from the lens of loss landscapes. Specifically, we visualize the loss landscape of the Kullback-Leibler (KL) divergence between node embeddings obtained before and after the model editing process in GNNs. We found that a slight weight perturbation can significantly enlarge the KL divergence. In contrast, other types of neural networks, such as Multi-Layer Perceptrons (MLPs), exhibit a much flatter region of the KL loss landscape and display greater robustness against weight variations. Such observations align with our viewpoint that after editing on misclassified samples, GNNs are prone to widely propagating the editing effect and affecting the remaining nodes.

Based on the sharp loss landscape of model editing in GNNs, we propose Editable Graph Neural Network (EGNN), a neighbor propagation-free approach to correct the model prediction on the graph data. Specifically, suppose we have a well-trained GNN and we aim to correct its prediction on some of the misclassified nodes. EGNN stitches a randomly initialized MLP to the trained GNN. We then train the MLP for a few iterations to ensure that it does not significantly alter the model's prediction. When performing the edit, we only update the parameter of the stitched MLP while freezing the parameter of GNNs during the model editing process. In particular, the node embeddings from GNNs are first inferred offline. Then MLP learns an additional representation, which is then combined with the fixed embeddings inferred from GNNs to make the final prediction. When a misclassified node is received, the gradient is back propagated to update the parameters of MLP instead of GNNs'. In this way, we decouple the *neighbor propagation process* of learning the structure-aware node embeddings from the *model editing process* of correcting the misclassified nodes. Thus, EGNN disables the propagation during editing while still utilizing the neighbor propagation scheme for node prediction to obtain satisfactory results. Compared to directly applying the existing model editing methods to GNNs:

- We can leverage the GNNs' structure learning meanwhile avoiding the spreading edition errors to guarantee the overall node classification task.
- The experimental results validate our solution which could address all the erroneous samples and deliver up to **90% improvement in overall accuracy**.
- Via freezing GNNs' part, EGNN is scalable to address misclassified nodes in the million-size graphs. We save more than 2× in terms of memory footprint and model editing time.

2 PRELIMINARY

Graph Neural Networks. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $\mathcal{V} = (v_1, \cdots, v_{|\mathcal{V}|})$ and $\mathcal{E} = (e_1, \cdots, e_{|\mathcal{E}|})$ being the set of nodes and edges, respectively. Let $X \in \mathbb{R}^{|\mathcal{V}| \times d}$ be the node feature matrix. $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the graph adjacency matrix, where $A_{i,j} = 1$ if $(v_i, v_j) \in \mathcal{E}$ else $A_{i,j} = 0$. $\tilde{A} = \tilde{D}^{-\frac{1}{2}}(A+I)\tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, where $\tilde{D}$ is the degree matrix of $A + I$. In this work, we are mostly interested in the task of node classification, where each node $v \in \mathcal{V}$ is associated with a label y_v, and the goal is to learn a representation h_v from which y_v can be easily predicted. To obtain such a representation, GNNs follow a neural message passing scheme [21]. Specifically, GNNs recursively update the representation of a node by aggregating representations of its neighbors. For example, the lth Graph Convolutional Network (GCN) layer [21] can be defined as:

$$H^{(l+1)} = \text{ReLU}(\tilde{A}H^{(l)}\Theta^{(l)}), \tag{1}$$

where $H^{(l)}$ is the node embedding matrix containing the h_v for each node v at the lth layer and $H^{(0)} = X$. $\Theta^{(l)}$ is the weight matrix of the lth layer.

The Model Editing Problem. The goal of model editing is to alter a base model's output for a misclassified sample x_e as well as its similar samples via model finetuning only using a single pair of input x_e and desired output y_e while leaving model behavior on unrelated inputs intact [15–17]. We are the first to propose the model editing problem in graph data, where the decision faults on a small number of critical nodes can lead to significant financial loss and/or fairness concerns. For the node classification, suppose a well-trained GNN incorrectly predicts a specific node. **Model editing** is used to correct the undesirable prediction behavior for that node by using the node's features and desired label to update the model. Ideally, the model editing ensures that the updated model makes accurate predictions for the specific node and its similar samples while maintaining the model's original behavior for the remaining unrelated inputs. Some model editors, such as the one presented in this paper, require a training phase before they can be used for editing.

3 PROPOSED METHODS

In this section, we first empirically show vanilla model editing performs extremely worse for GNNs compared with MLPs due to node propagation (Section 3.1). Intuitively, due to the message-passing mechanism in GNNs, editing the model behavior on a single node can propagate changes across the entire graph, significantly altering the node's original representation. Then through visualizing

Table 1: The test accuracy (%) before ("w./o. edit") and after editing ("w./ edit") on one single data point. Δ Acc is the accuracy drop before and after performing the edit. All results are averaged over 50 simultaneous model edits. The best result is highlighted by bold faces.

		GCN	GraphSAGE	MLP
	w./o. edit	**89.4**	86.6	71.8
Cora	w./ edit	**84.36**	82.06	68.33
	Δ Acc.	5.03↓	4.53↓	**3.46 ↓**
	w./o. edit	**51.19**	49.03	46.77
Flickr	w./ edit	13.94	17.15	**36.68**
	Δ Acc.	37.25↓	31.88↓	**10.08 ↓**
	w./o. edit	95.52	**96.55**	72.41
Reddit	w./ edit	**75.20**	55.85	69.86
	Δ Acc.	20.32↓	40.70↓	**2.54 ↓**
	w./o. edit	**70.20**	68.38	52.65
ogbn-arxiv	w./ edit	23.70	19.06	**45.15**
	Δ Acc.	46.49↓	49.31↓	**7.52↓**

the loss landscape, we found that for GNNs, even a slight weight perturbation, the node representation will be far away from the original one (Section 3.2). Based on the observation, we propose a propagation-free GNN editing method called EGNN (Section 3.3).

3.1 Motivation: Model Editing may Cry in GNNs

Setting: We train GCN, GraphSAGE, and MLP on Cora, Flickr, Reddit, and ogbn-arxiv, respectively, following the training setup as described in Section 5. To evaluate the difficulty of editing, *we ensured that the node to be edited was not present during training*, meaning that *the models were trained inductively*. Specifically, we trained the model on a subgraph containing only the training node and evaluated its performance on the validation and test set of nodes. Next, we selected a misclassified node from the validation set and applied gradient descent only on that node until the model made a correct prediction for it. Following previous work [15, 17], we perform 50 independent edits and report the averaged test accuracy before and after performing a single edit.

Results: As shown in Table 1, we observe that **(1)** GNNs consistently outperform MLP on all the graph datasets before editing. This is consistent with the previous graph analysis results, where the neural message passing involved in GNNs extracts the graph topology to benefit the node representation learning and thereby the classification accuracy. **(2)** After editing, the accuracy drop of GNNs is significantly larger than that of MLP. For example, Graph-SAGE has an almost 50% drop in test accuracy on ogbn-arxiv after editing even a single point. MLP with editing even delivers higher overall accuracies on Flickr and ogbn-arxiv compared with GNN-based approaches. One of the intuitive explanations is the slightly fine-tuned weights in MLP mainly affect the target node, instead of other unrelated samples. However, due to the message-passing mechanism in GNNs, the edited node representation can be propagated over the whole graph and thus change the decisions on a large area of nodes. These comparison results reveal the unique challenge in editing the correlated nodes with GNNs, compared with the conventional neural networks working on isolated samples. **(3)** After editing, the test accuracy of GCN, GraphSAGE, and

MLP become too low to be practically deployed. This is quite different to the model editing problems in computer vision and natural language processing, where the modified models only suffer an acceptable accuracy drop.

3.2 Sharp Locality of GNNs through Loss Landscape

Intuitively, due to the message-passing mechanism in GNNs, editing the model behavior for a single node can cause the editing effect to propagate across the entire graph. This propagation pushes the node representation far from its original one. **Thus, we hypothesized that the difficulty in editing GNNs as being due to the neighbor propagation of GNNs.** The model editing aims to correct the prediction of the misclassified node using the cross-entropy loss of desired label. Intuitively, the large accuracy drop can be interpreted as the low model prediction similarity before and after model editing, named as the locality.

To quantitatively measure the locality, we use the metric of KL divergence between the node representations learned before and after model editing. The higher KL divergence means after editing, the node representation is far away from the original one. In other words, the higher KL divergence implies poor model locality, which is undesirable in the context of model editing. Particularly, we visualize the locality loss landscape for Cora dataset in Figure 1. We observe several **insights:** (1) GNNs (e.g., GCN and GraphSAGE) suffer from a much sharper loss landscape. Even slightly editing the weights, KL divergence loss is dramatically enhanced. That means GNNs are hard to be fine-tuned while keeping the locality. (2) MLP shows a flatter loss landscape and demonstrates much better locality to preserve overall node representations. This is consistent to the accuracy analysis in Table 1, where the accuracy drop of MLP is smaller.

3.3 EGNN Neighbor Propagation Free GNN Editing

In our previous analysis, we hypothesized that the difficulty in editing GNNs as being due to the neighbor propagation. However, as Table 1 suggested, the neighbor propagation is necessary for obtaining good performance on graph datasets. On the other hand, MLP could stabilize most of the node representations during model editing although it has worse node classification capability. Thus, we need to find a way to "disable" the propagation during editing while still utilizing the neighbor propagation scheme for node prediction to obtain satisfactory results. Following the motivation, we propose to combine a compact MLP to the well-trained GNN and only modify the MLP during editing. In this way, we can correct the model's predictions through this additional MLP while freezing the neighbor propagation. Meanwhile during inference, both the GNN and MLP are used together for prediction in tandem to harness the full potential of GNNs for prediction.

Before editing. We first stitch a randomly initialized compact MLP to the trained GNN. To mitigate the potential impact of random initialization on the model's prediction, we introduce a training procedure for the stitched MLP, as outlined in Algorithm **??** "MLP TRAINING PROCEDURE": we train the MLP for a few iterations to ensure that it does not significantly alter the model's prediction. By
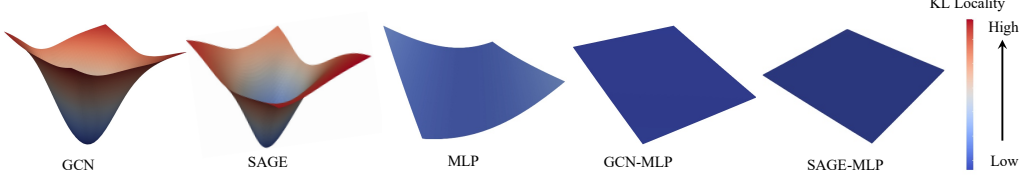
Figure 1: The loss landscape of various model architectures on Cora dataset. Similar results can be found in Appendix C
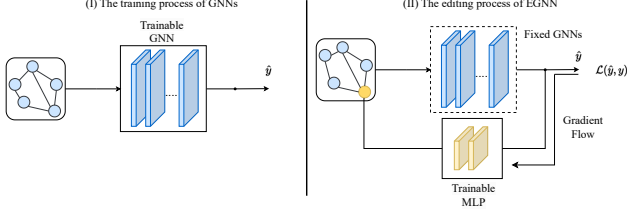

Figure 2: The overview of EGNN . We fix the backbone of GNNs (in blue), while only update the small MLPs (in orange) during editing. The wrongly predicted nodes are highlighted with orange color, and the MLP only require its node feature.

freezing GNN's weights, we first get the node embedding h_v at the last layer of the trained GNN by running a single forward pass. We then stitch the MLP with the trained GNNs. Mathematically, we denote the MLP as g_Φ where Φ is the parameters of MLP. For a given input sample x_v, y_v, the model output now becomes $h_v + g_\Phi(x_v)$. We calculate two loss based on the prediction, i.e., the task-specific loss $\mathcal{L}_{task}$ and the locality loss $\mathcal{L}_{loc}$. Namely,

$$\mathcal{L}_{task} = -\log p_\Phi(y_v|h_v + g_\Phi(x_v)),$$
$$\mathcal{L}_{loc} = KL(h_v + g_\Phi(x_v)||h_v),$$

where $h_v + g_\Phi(x_v)$ is the model prediction with the additional MLP and $p_\Phi(y_v|h_v + g_\Phi(x_v))$ is the probability of class y_v given by the model. $\mathcal{L}_{task}$ is the cross-entropy between the model prediction and label. $\mathcal{L}_{loc}$ is the locality loss, which equals KL divergence between the original prediction h_v and the prediction with the additional MLP $h_v + g_\Phi(x_v)$. The final loss $\mathcal{L}$ is the weighted combination of two parts, i.e., $\mathcal{L} = \mathcal{L}_{task} + \alpha \mathcal{L}_{loc}$ where α is the weight for the locality loss. $\mathcal{L}$ is used to guide the MLP to fit the task while keep the model prediction unchanged.

When editing. EGNN freezes the model parameters of GNN and only updates the parameters of MLP. Specifically, as outlined in Algorithm **??** "EGNN EDIT PROCEDURE", we update the parameters of MLP until the model prediction for the misclassified sample is corrected. Since MLP only relies on the node features, we can easily perform these updates in mini-batches, which enables us to edit GNNs on large graphs.

Lastly, we visualize the KL locality loss landscape of EGNN (including GCN-MLP and SAGE-MLP) in Figure 1. It is seen that the proposed EGNN shows the most flattened loss landscape than MLP and GNNs, which implied that EGNN can preserve overall node representations better than other model architectures.

4 RELATED WORK AND DISCUSSION

Due to the page limit, below we discuss the related work on model editing. We also discuss the limitation in Appendix B.

Model Editing. Many approaches have been proposed for model editing. The most straightforward method adopts standard finetuning to update model parameters based on misclassified samples while preserving model locality via constraining parameters travel distance in model weight space [22, 23]. Work [24] introduces meta-learning to find a pre-trained model with rapid and easy finetuned ability for model editing. Another way to facilitate model editing relies on external learned editors to modify model editing considering several constraints [16–18, 25]. The editing of the activation map is proposed to correct misclassified samples in [26, 27] due to the belief of knowledge attributed to model neurons. While all these works either update base model parameters or import external separate modules for model prediction to induce desired prediction change, the considered data is i.i.d. and may not work well in graph data due to essential node interaction during neighborhood propagation. In this paper, we propose EGNN, using a stitched MLP module to edit the output space of the base GNN model, for node classification tasks. The key insight behind this solution is the sharp locality of GNNs, i.e., the prediction of GNNs can be easily altered after model editing.

5 EXPERIMENTS

The experiments are designed to answer the following research questions. **RQ1:** Can EGNN correct the wrong model prediction? Moreover, what is the difference in accuracy before and after editing using EGNN ? **RQ2:** Can the edits generalize to correct the model prediction on other similar inputs? **RQ3:** What is the time and memory requirement of EGNN to perform the edits?

5.1 Experimental Setup

Datasets and Models. To evaluate EGNN , we adopt four small-scale and four large-scale graph benchmarks from different domains. For small-scale datasets, we adopt Cora, A-computers [28], A-photo [28], and Coauthor-CS [28]. For large-scale datasets, we adopt Reddit [2], Flickr [4], *ogbn-arxiv* [5], and *ogbn-products* [5]. We integrate EGNN with two popular models: GCN [21] and GraphSAGE [2]. *To avoid creating confusion, GCN and GraphSAGE are all trained with the whole graph at each step.* We evaluate EGNN under the **inductive setting**. Namely, we trained the model on a subgraph containing only the training node and evaluated it on the whole graph. Details about the hyperparameters and datasets are in Appendix A.

Compared Methods. We compare our EGNN editor with the following two baselines: the vanilla gradient descent editor (GD) and Editable Neural Network editor (ENN) [15]. GD is the same editor

Table 2: The results on four small scale datasets after applying one single edit. The reported number is averaged over 50 independent edits. SR is the edit success rate, Acc is the test accuracy after editing, and DD are the test drawdown, respectively. "OOM" is the out-of-memory error.

|	Editor	Cora			A-computers			A-photo			Coauthor-CS		
		Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑
GCN	GD	84.37±5.84	5.03±6.40	1.0	44.78±22.41	43.09±22.32	1.0	28.70±21.26	65.08±20.13	1.0	91.07±3.23	3.30±2.22	1.0
	ENN	37.16±3.80	52.24±4.76	1.0	15.51±10.99	72.36±10.87	1.0	16.71±14.81	77.07±15.20	1.0	4.94±3.78	89.43±3.34	1.0
	EGNN	**87.80**±2.34	**1.80**±2.13	**1.0**	**82.85**±5.20	**2.32**±5.11	0.98	**91.97**±5.85	**2.39**±5.34	**1.0**	**94.54**±0.07	**-0.17**±0.07	**1.0**
Graph-SAGE	GD	82.06±4.33	4.54±5.32	1.0	21.68±20.98	61.15±20.33	1.0	38.98±30.24	55.32±29.35	1.0	90.15±5.58	5.01±5.32	1.0
	ENN	33.16±1.45	53.44±2.23	1.0	16.89±16.98	65.94±16.75	1.0	15.06±11.92	79.24±11.25	1.0	13.71±2.73	81.45±2.11	1.0
	EGNN	**85.65**±2.23	**0.55**±1.26	**1.0**	**84.34**±4.84	**2.72**±5.03	0.94	**92.53**±2.90	**1.83**±3.22	**1.0**	**95.27**±0.08	**-0.01**±0.10	**1.0**
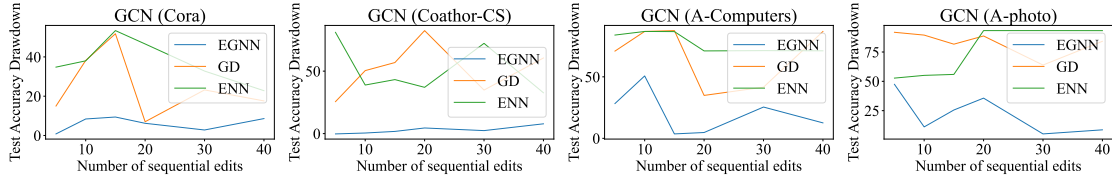
Figure 3: Sequential edit drawdown of GCN on four small scale datasets.
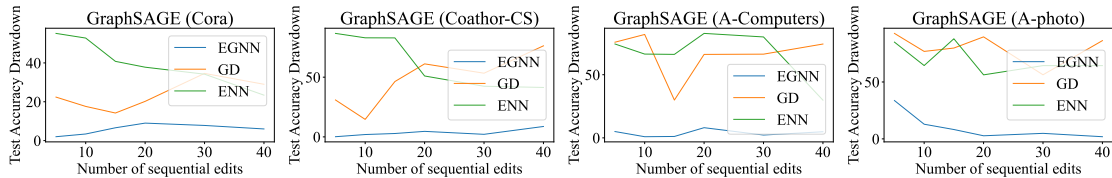

Figure 4: Sequential edit drawdown of GraphSAGE on four small scale datasets.
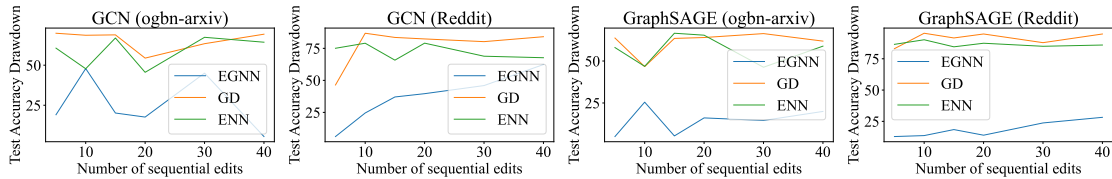

Figure 5: Sequential edit test drawdown of GCN and GraphSAGE on Reddit and ogbn-arxiv dataset.

we used in our preliminary analysis in Section 3. **We note that for other model editing, e.g., MEND [16], SERAC [17] are tailored for NLP applications, which cannot be directly applied to the graph area**. Specifically, GD applies the gradient descent **on the parameters of GNN** until the GNN makes right prediction. ENN trains **the parameters of GNN** for a few steps to make it prepare for the following edits. Then similar to GD editor, it applies the gradient descent **on the parameters of GNN** until the GNN makes right prediction. For EGNN , we only train **the stitched MLP** for a few steps. Then we only update **weights of MLP** during edits. Detailed hyperparameters are listed in Appendix A.

Evaluation Metrics. Following previous work [15–17], we evaluate the effectiveness of different methods by the following three metrics. **DrawDown (DD)**, which is the mean absolute difference of test accuracy before and after performing an edit. A smaller drawdown indicates a better editor locality. **Success Rate (SR)**, which is defined as the rate of edits, where the editor successfully corrects the model prediction. **Edit Time**, which is defined as the wall-clock time of a single edit that corrects the model prediction.

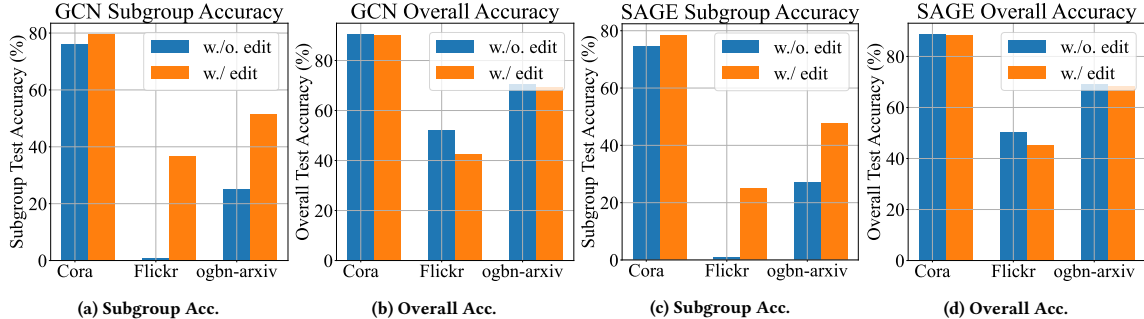## 5.2 The Effectiveness of EGNNn Editing GNNs

In many real-world applications, it is common to encounter situations where our trained model produces incorrect predictions on unseen data. It is crucial to address these errors as soon as they are identified. To assess the usage of editors in real-world applications (**RQ1**), **we select misclassified nodes from the validation set, which is not seen during the training process.** Then we employ the editor to correct the model's predictions for those misclassified nodes, and measure the drawdown and edit success rate on the test set.

The results after editing on a single node are shown in Table 2 and Table 3. We observe that

❶ *Unlike editing Transformers on text data [16, 17, 29], all editors can successfully correct the model prediction in graph domain.* As shown in Table 3, all editors have 100% success rate when edit GNNs. In contrast, for transformers, the edit success rate is often less than 50% and drawdown is much smaller than GNNs [16, 17, 29]. This observation suggests that **unlike transformers, GNNs can be easily perturbed to produce correct predictions. However, at the cost of huge drawdown on other unrelated nodes. Thus,**

Table 3: The results on four large scale datasets after applying one single edit. "OOM" is the out-of-memory error.

	Editor	Flickr			Reddit			ogbn-arxiv			ogbn-products		
		Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑	Acc↑	DD↓	SR↑
GCN	GD	13.95±11.0	37.25±10.2	1.0	75.20±12.3	20.32±11.3	1.0	23.71±16.9	46.50±14.9	1.0	OOM	OOM	0
	ENN	25.82±14.9	25.38±16.9	1.0	11.16±5.1	84.36±3.1	1.0	16.59±7.7	53.62±6.7	1.0	OOM	OOM	0
	EGNN	**44.91**±12.2	**6.34**±10.3	**1.0**	**94.46**±0.4	**1.03**±0.6	**1.0**	**67.34**±8.7	**2.67**±4.4	**1.0**	**74.19**±3.4	**0.81**±0.23	**1.0**
Graph-SAGE	GD	17.16±12.2	31.88±12.2	1.0	55.85±22.5	40.71±20.3	1.0	19.07±14.1	36.68±10.1	1.0	OOM	OOM	0
	ENN	28.73±5.6	20.31±5.6	1.0	5.88±3.9	90.68±4.3	1.0	8.14±8.6	47.61±7.6	1.0	OOM	OOM	0
	EGNN	**43.52**±10.8	**5.12**±10.8	**1.0**	**96.50**±0.1	**0.05**±0.1	**1.0**	**67.91**±2.9	**0.64**±2.3	**1.0**	**76.27**±0.6	**0.17**±0.10	**1.0**

(a) Subgroup Acc.   (b) Overall Acc.   (c) Subgroup Acc.   (d) Overall Acc.

Figure 6: The subgroup and overall test accuracy before and after one single edit. The results are averaged over 50 independent edits.

the main challenge lies in maintaining the locality between predictions for unrelated nodes before and after editing. This observation aligns with our initial analysis, which highlighted the interconnected nature of nodes and the edit on a single node may propagate throughout the entire graph.

❷ EGNN *significantly outperforms both GD and ENN in terms of the test drawdown.* This is mainly because both GD and ENN try to correct the model's predictions by updating the parameters of Graph Neural Networks (GNNs). This process inevitably relies on neighbor propagation. In contrast, EGNN has much better test accuracy after editing. Notably, for Reddit, the accuracy drop decreases from roughly 80% to ≈ 1%, which is significantly better than the baseline. This is because EGNN decouples the neighbor propagation with the editing process. Interestingly, ENN is significantly worse than the vanilla editor, i.e., GD, when applied to GNNs. As shown in Appendix C, we found that this discrepancy arises from the ENN training procedure, which significantly compromises the model's performance to prepare it for editing.

In Figure 3, 4, and 5 we present the ablation study under the sequential setting. This is a more challenging scenario where the model is edited sequentially as errors arise. In particular, we plot the test accuracy drawdown against the number of sequential edits for GraphSAGE on the ogbn-arxiv dataset. We observe that ❸ EGNN *consistently surpasses both GD and ENN in the sequential setting.* However, the drawdown is considerably greater than that in the single edit setting. For instance, EGNN exhibits a 0.64% drawdown for GraphSAGE on the ogbn-arxiv dataset in the single edit setting, which escalates up to a 20% drawdown in the sequential edit setting. These results also highlight the hardness of maintaining the locality of GNN prediction after editing.

(a) Before Edit   (b) After Edit
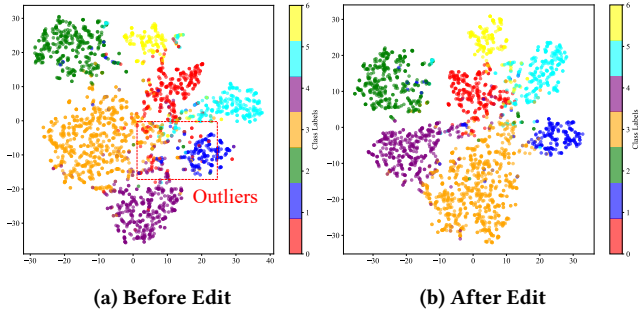
Figure 7: T-SNE visualizations of GNN embeddings before and after edits on the Cora dataset. The flipped nodes are all from class 0, which is marked in red color.

5.3 The Generalization of the Edits of EGNN

Ideally, we aim for the edit applied to a specific node to generalize to similar nodes while preserving the model's initial behavior for unrelated nodes. To evaluate the generalization of the EGNN edits, we conduct the following experiment:

(1) We first select a particular group (i.e., class) of nodes based on their labels. **(2)** Next, we randomly flip the labels of 10% of the training nodes within this group and train a GNN on the modified training set. **(3)** For each flipped training node, we correct the trained model's prediction for that node back to its original class and assess whether the model's predictions for other nodes in the same group are also corrected. If the model's predictions for other nodes in the same class are also corrected after modifying a single flipped node, it indicates that the EGNN edits can effectively generalize to address similar erroneous behavior in the model.

Table 4: The edit time and memory required for editing.

	Editor	Flickr		Reddit		ogbn-arxiv		ogbn-products	
		Edit Time (ms)	Peak Memory (MB)	Edit Time (ms)	Peak Memory (MB)	Edit Time (ms)	Peak Memory (MB)	Edit Time (ms)	Peak Memory (MB)
GCN	GD	379.86	707	1835.24	3429	663.17	967	OOM	OOM
	EGNN	246.63	315	765.15	2089	299.71	248	5122.53	5747
Graph-SAGE	GD	712.07	986	4781.92	5057	668.77	1109	OOM	OOM
	EGNN	389.37	328	1516.68	2252	174.82	260	5889.59	6223

To answer **RQ2**, we conduct the above experiments and report the **subgroup and overall test accuracy** after performing a single edit on the flipped training node. The results are shown in Figure 6. We observe that: ❹ *From Figure 6a and Figure 6c,* EGNN *significantly improves the subgroup accuracy after performing even a single edit.* Notably, the subgroup accuracy is significantly lower than the overall accuracy. For example, on Flickr dataset, both GCN and GraphSAGE have a subgroup accuracy of less than 5% before editing. This is mainly because the GNN is trained on the graph where 10% labels of the training node in the subgroup are flipped. However, even after editing on a single node, the subgroup accuracy is significantly boosted. These results indicate that the EGNN edits can effectively generalize to address the wrong prediction on other nodes in the same group. In Figure 7, we also visualize the node embeddings before and after editing by EGNN on the Cora dataset. We note that all of the flipped nodes are from class 0, which is marked in red color in Figure 7. Before editing, the red cluster has many outliers that lie in the embedding space of other classes. This is mainly because the labels of some of the nodes in this class are flipped. In contrast, after editing, the nodes in the red cluster become significantly closer to each other, with a substantial reduction in the number of outliers.

5.4 The Efficiency of EGNN

We want to patch the model as soon as possible to correct errors as they appear. Thus ideally, the editor should be efficient and scalable to large graphs. Here we summarize the edit time and memory required for performing the edits in Table 4. We observe that EGNN is about $2 \sim 5\times$ faster than the GD editor in terms of the wall-clock edit time. This is because EGNN only updates the parameters of MLP, and totally gets rid of the expensive graph-based sparse operations [19, 20, 30]. Also, updating the parameters of GNNs requires storing the node embeddings in memory, which is directly proportional to the number of nodes in the graph and can be exceedingly expensive for large graphs. However, with EGNN , we only use node features for updating MLPs, meaning that memory consumption is not dependent on the graph size. Consequently, EGNN can efficiently scale up to handle graphs with millions of nodes, e.g., ogbn-products, whereas the vanilla editor raises an OOM error.

6 CONCLUSION

In this paper, we explore a and important problem, i.g., GNNs model editing for node classification. We first empirically observe that the vanilla model editing method may not perform well due to node aggregation, and then theoretically investigate the underlying

reason through the lens of locality loss landscape with quantitative analysis. Furthermore, we propose EGNN to correct misclassified samples while preserving other intact nodes, via stitching a trainable MLP. In this way, the power of GNNs for prediction and the editing-friendly MLP can be integrated together in EGNN.

REFERENCES

[1] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[2] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[3] Hongyi Ling, Zhimeng Jiang, Youzhi Luo, Shuiwang Ji, and Na Zou. Learning fair graph representations via automated data augmentations. In *International Conference on Learning Representations*, 2023.

[4] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

[5] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[6] Kaixiong Zhou, Zirui Liu, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Table2graph: Transforming tabular data to unified weighted graph.

[7] Zhimeng Jiang, Xiaotian Han, Chao Fan, Zirui Liu, Na Zou, Ali Mostafavi, and Xia Hu. Fmp: Toward fair graph message passing against topology bias. *arXiv preprint arXiv:2202.04187*, 2022.

[8] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, Qingquan Song, Jundong Li, and Xia Hu. Geometric graph representation learning via maximizing rate reduction. In *Proceedings of the ACM Web Conference 2022*, pages 1226–1237, 2022.

[9] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In *International Conference on Machine Learning*, pages 8230–8248. PMLR, 2022.

[10] Hongyi Ling, Zhimeng Jiang, Meng Liu, Shuiwang Ji, and Na Zou. Graph mixup with soft alignments. In *International Conference on Machine Learning*. PMLR, 2023.

[11] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *NeurIPS*, 2022.

[12] Kaixiong Zhou, Ninghao Liu, Fan Yang, Zirui Liu, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Adaptive label smoothing to regularize large-scale graph training. *CoRR*, abs/2108.13555, 2021.

[13] Daniele Petrone and Vito Latora. A dynamic approach merging network theory and credit risk techniques to assess systemic risk in financial networks. *Scientific Reports*, 8(1):5561, 2018.

[14] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, 19(1):22–36, 2017.

[15] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. *arXiv preprint arXiv:2004.00345*, 2020.

[16] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

[17] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR, 2022.

[18] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. *arXiv preprint arXiv:2104.08164*, 2021.

[19] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2022.

[20] Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. MLPInit: Embarrassingly simple GNN training acceleration with MLP initialization. In *The Eleventh International Conference on Learning Representations*, 2023.

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[22] Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. Modifying memories in transformer models. *arXiv preprint arXiv:2012.00363*, 2020.

[23] Matthew Sotoudeh and A Thakur. Correcting deep neural networks with small, generalizing patches. In *Workshop on safety and robustness in decision making*, 2019.

[24] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. *arXiv preprint arXiv:2004.00345*, 2020.

[25] Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. Do language models have beliefs? methods for detecting, updating, and visualizing model beliefs. *arXiv preprint arXiv:2111.13654*, 2021.

[26] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. *arXiv preprint arXiv:2104.08696*, 2021.

[27] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

[28] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[29] Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. Transformer-patcher: One mistake worth one neuron. In *The Eleventh International Conference on Learning Representations*, 2023.

[30] Zirui Liu, Shengyuan Chen, Kaixiong Zhou, Daochen Zha, Xiao Huang, and Xia Hu. Rsc: Accelerating graph neural networks training via randomized sparse computations. *arXiv preprint arXiv:2210.10737*, 2022.

[31] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[32] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[33] Zirui Liu, Qingquan Song, Kaixiong Zhou, Ting-Hsiang Wang, Ying Shan, and Xia Hu. Towards interaction detection using topological analysis on neural networks. *CoRR*, abs/2010.13015, 2020.

[34] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[35] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

[36] Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In *Privacy, Security, and Trust in KDD: First ACM SIGKDD International Workshop, PinKDD 2007, San Jose, CA, USA, August 12, 2007, Revised Selected Papers*, pages 153–171. Springer, 2008.

[37] Zhimeng Jiang, Xiaotian Han, Hongye Jin, Guanchu Wang, Na Zou, and Xia Hu. Weight perturbation can help fairness under distribution shift. *arXiv preprint arXiv:2303.03300*, 2023.

[38] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020.

[39] Enyan Dai and Suhang Wang. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 680–688, 2021.

[40] Zhimeng Jiang, Xiaotian Han, Chao Fan, Fan Yang, Ali Mostafavi, and Xia Hu. Generalized demographic parity for group fairness. In *International Conference on Learning Representations*, 2022.

[41] Xiaotian Han, Zhimeng Jiang, Hongye Jin, Zirui Liu, Na Zou, Qifan Wang, and Xia Hu. Retiring ΔDP: New distribution-level metrics for demographic parity. *arXiv preprint arXiv:2301.13443*, 2023.

[42] Ruixiang Tang, Mengnan Du, Yuening Li, Zirui Liu, and Xia Hu. Mitigating gender bias in captioning systems. *CoRR*, abs/2006.08315, 2020.

A EXPERIMENTAL SETTING

A.1 Datasets for node classification

The statistical information of datasets are shown in Table 5.

Table 5: Statistics for datasets used for node classification.

Dataset	# Nodes.	# Edges	# Classes	# Feat	Density
Cora	2,485	5,069	7	1433	0.72‰
A-computers	13,381	245,778	10	767	2.6‰
A-photo	7,487	119,	8	745	4.07‰
Coauthor-CS	18,333	81,894	15	6805	0.49‰
Flickr	89,250	899,756	7	500	0.11‰
Reddit	232,965	23,213,838	41	602	0.43‰
ogbn-arxiv	169,343	1,166,243	40	128	0.04‰
ogbn-products	2,449,029	61,859,140	47	218	0.01‰

B LIMITATIONS AND FUTURE WORK

Despite that EGNN is effective, generalized, and efficient, its main limitation is that currently it will incur the larger inference latency, due to the extra MLP module. However, we note that this inference overhead is negligible. This is mainly because the computation of MLP only involves dense matrix operation, which is way more faster than the graph-based sparse operation. The future work comprises several research directions, including (1) Enhancing the efficiency of editable graph neural networks training through various perspectives (e.g., model initialization, data, and gradient); (2) understanding why vanilla editable graph neural networks training fails from other perspectives (e.g., interpretation and information bottleneck) [31–33]; (3) Advancing the scalability, speed, and memory efficiency of editable graph neural networks training [19, 20, 30]; (4) Expanding the scope of editable training for other tasks (e.g., link prediction, and knowledge graph) [34, 35]; (5) Investigating the potential issue concerning privacy, robustness, and fairness in the context of editable graph neural networks training [36–42].

C MORE EXPERIMENTAL RESULTS

C.1 More Loss Landscape Results

We visualize the locality loss landscape for Flickr dataset in Figure 8. Similarly, Z axis denotes the KL divergence, X-Y axis is centered on the original model weights before editing and quantifies the weight perturbation scale after model editing. We observe similar observations: (1) GNNs architectures (e.g., GCN and GraphSAGE) suffer from a much sharper loss landscape at the convergence of original model weights. KL divergence locality loss is dramatically enhanced even for slight weights editing. (2) MLP shows a flatter loss landscape and demonstrates mild locality to preserve overall node representations, which is consistent with the accuracy analysis in Table 1. (3) The proposed EGNN shows the most flattened loss landscape than MLP and GNNs, which implied that EGNN can preserve overall node representations better than other model architectures.
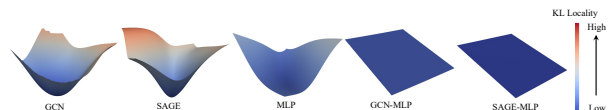

Figure 8: The loss landscape of various GNNs architectures on Flickr dataset.