GNNs Also Deserve Editing, and They Need It More Than Once

Shaochen (Henry) Zhong [*1] **Duy Le** [*2] **Zirui Liu** [1] **Zhimeng Jiang** [3] **Andrew Ye** [2] **Jiamu Zhang** [2] **Jiayi Yuan** [1] **Kaixiong Zhou** [4] **Zhaozhuo Xu** [5] **Jing Ma** [2] **Shuai Xu** [2] **Vipin Chaudhary** [2] **Xia Hu** [1]

Abstract

Suppose a self-driving car is crashing into pedestrians, or a chatbot is instructing its users to conduct criminal wrongdoing; the stakeholders of such products will undoubtedly want to patch these catastrophic errors as soon as possible. To address such concerns, *Model Editing:* the study of efficiently patching model behaviors without significantly altering their general performance, has seen considerable activity, with hundreds of editing techniques developed in various domains such as CV and NLP. However, **the graph learning community has objectively fallen behind with only a few Graph Neural Network-compatible — and just one GNN-specific — model editing methods available**, where all of which are limited in their practical scope. We argue that the impracticality of these methods lies in their lack of *Sequential Editing Robustness*: the ability to edit multiple errors sequentially, and therefore fall short in effectiveness, as this setup mirrors how errors are discovered and addressed in the real world. In this paper, we delve into the specific reasons behind the difficulty of editing GNNs in succession and observe the root cause to be model overfitting. We subsequently propose a simple yet effective solution — SEED-GNN — by leveraging overfit-prevention techniques in a GNN-specific context to derive the first and only GNN model editing method that scales practically. Additionally, we formally frame the task paradigm of GNN editing and hope to inspire future research in this crucial but currently overlooked field. Please refer to our GitHub repository for code and checkpoints.

[*]Equal contribution [1]Rice University [2]Case Western Reserve University [3]Texas A&M University [4]North Carolina State University [5]Stevens Institute of Technology. Correspondence to: Shaochen (Henry) Zhong <shaochen.zhong@rice.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

When training and evaluating models, comprehensive metrics like classification accuracy are regarded as some of the most pursued and crucial criteria for machine learning. Indeed, even a slight but consistent improvement in these metrics often transfers to immediate practical gain when these systems are deployed. Yet, metrics that evaluate such models at large critically lack the capability to effectively address the particular errors that these same models may potentially accrue over their training period. Simply classifying outputs to a degree of correctness is not enough, since **two wrong outputs that appear the same on paper can result in drastically different impacts in the real world**, as some mistakes are inherently more damaging than others. Take, for instance, a self-driving car that wrongfully classifies a sedan as an SUV versus another one that wrongfully classifies a street-crossing child as a bunny — while both mistakes are considered just "one wrong prediction" under metrics like overall accuracy, the latter one is undoubtedly more catastrophic.

Model Editing — or Knowledge Editing — aims to address this oversight by studying the efficient patching of particular undesirable model outputs without significantly altering the model's general behavior on unrelated input (Yao et al., 2023; Gu et al., 2023). The approach has since been widely adopted due to its ability to efficiently deliver (ideally guaranteed) patches on errors that models may make. Furthermore, **editing techniques often carry great practical significance due to their relevance in model development and maintenance**, as high-profile failure cases will almost always occur during commercial deployment and after initial development (e.g., training, fine-tuning, and internal testing). As such, the ability to efficiently address post-hoc high-profile failure cases without expensive and perhaps unrealistic remedies — such as a complete retraining of the model — is not only valuable but, in some cases, the only practical way to correct model behavior.

1.1. GNNs Deserve Model Editing Too, but Why Don't They Get Any?

Despite the rapid advancement of model editing in domains like Computer Vision (CV) (Sotoudeh & Thakur, 2021;

Santurkar et al., 2021; Tanno et al., 2022), Natural Language Processing (NLP) (Meng et al., 2023b;a; Zhu et al., 2020; Cao et al., 2021; Mitchell et al., 2022), and others (Mazzia et al., 2023; Cheng et al., 2023; Yu et al., 2023), the graph learning community has largely been left behind, with EGNN by Liu et al. (2023a) being the only Graph Neural Network (GNN) specific model editing technique proposed. The considerable difference in popularity between the graph and non-graph domains in terms of model editing techniques begs the intriguing and natural question — Why?

Although there is no absolute answer to this question, as it depends on different researchers' tastes, resources, expertise, and exposure, we believe the measurable root of the phenomenon to be mainly tri-fold:

Lack of recognition, as graphs are less intuitive than text and images. On the mainstream data types (e.g., text and image) with well-developed model editing techniques, there are various straightforward scenarios in which incorrect model outputs may pose severe consequences, like the aforementioned self-driving example. However, in common graph-related scenarios such as social networks or recommender systems, such catastrophic outcomes are not as frequently encountered in day-to-day life, nor as "graphic" or "intuitive" in comparison to catastrophic failure cases in text and image space. This may lead people to overlook the research on Graph/GNN model editing.

Nonetheless, we note that **graph-based learning certainly has high-impact applications**, such as in loan approval (Petrone & Latora, 2018), the detection of fake news (Shu et al., 2017), crime prevention (Wang et al., 2022), substance synthesis (Yu et al., 2022), drug repurpose (Hsieh et al., 2021), and health diagnosis (Li et al., 2022). For example, mistakes in the prediction of therapeutic drugs can lead to the waste of resources during wet lab testing, delaying the discovery of novel therapies and worsening patient conditions. Similar severity can be argued should a graph-based crime prevention model make wrong predictions about potential suspects, resulting in unpredictable high-stress police-civilian interaction without proper merit.

Lack of practical usability, as current GNN-compatible editing techniques all exhibit inadequate *Sequential Editing Robustness*. Even the most successful GNN editing method currently available (EGNN by Liu et al. (2023a)) can only afford to fix a single error or, if lucky, at most a few (< 5) instances sequentially before suffering unacceptable performance drops (EGNN may induce up to 49.3% drop on overall accuracy upon multiple edits). Such a decreased level of performance cannot support any form of practical utilization, as those high-profile failure cases mentioned above will always manifest in a sequential manner, where **there is certainly more than one failure case required to be taken care of.** To make things worse, the sequential nature of error discovery implies a later occurred error case might be the direct result of previously conducted edits; yet, some earlier patched errors might reemerge due to the overwrite effect of an inconsiderate later patch. Thus, we define having strong Sequential Editing Robustness as the ability to sequentially patch a large number of editing targets without suffering in general performance; and we consider it to be one of the most reflective metrics for gauging the practical usability of an editing method.

GNNs are intrinsically less robust to editing influence due to neighbor aggregation. As revealed in Liu et al. (2023a), the neighborhood aggregation mechanism in GNNs will propagate the changes made by the model editing on one node to its neighboring nodes, or even the whole graph, which can easily degrade the overall prediction performance of the model, making GNN an editing-sensitive architecture by nature. More on this in Section 4.1.

1.2. The Enemy is, Once Again, Overfitting

As mentioned in Section 1.1, GNN editing lacks both recognition and practical performance, since it is intrinsically a difficult task, resulting in the area of GNN model editing remaining largely unexplored. In this work, we reveal that **the main reason that GNNs are editing-sensitive is** not solely due to neighborhood aggregation (or message passing) as suggested by Liu et al. (2023a), but lies more in **the simple overfitting of editing targets**. To mitigate this issue, we leverage a vanilla batching operation to stitch each editing target at hand with a subset of training samples and previously edited targets, resulting in edited models that generate correct outputs and exhibit greater generalization capabilities. Our main claims and contributions are:

- **The first sequential and robust graph editing method.** We propose Sequentially Editable Graph Neural Networks, or SEED-GNN, which employs several lightweight operations to mitigate overfitting and integrates seamlessly with existing editing pipelines (Liu et al., 2023a). We find that our method performs vastly better, is more scalable, and remains generalized in contrast to all existing graph-editing techniques under all evaluations we conducted (and we evaluate plenty comprehensively).
- **A better understanding of why GNNs are edit-sensitive.** In addition to our proposed method, we additionally provide an in-depth investigation into why graph neural networks react to edits so severely, and determine the primary cause to be the overfitting of edit targets. This conclusion subverts the investigation of prior art (Liu et al., 2023a), opening a new potential direction of GNN editing research.
- **The comprehensive formalization/evaluation of the**

task of GNN editing. Finally, we formulate a rather extensive set of descriptions, objectives, and evaluation metrics specific to the task of (sequential) graph editing for the purpose of promoting and aiding future research in this field. We also replicated and evaluated almost all visible GNN editing methods — as well as several intuitive adaptations of finetuning techniques — against these metrics, and we plan to release all model checkpoints and code implementation in our GitHub repository to provide our fellow scholars with a friendly workbench.

Moreover, we must note the field of GNN model editing is a field with one of the highest degrees of freedom, which is often appreciated by many ML scholars. This is because editing experiments are often extremely fast to run, yet it is no longer bonded within the typical train-validation-evaluation pipeline (which is especially exciting due to the connected nature of graph data), forming a perfect playground for technical creativity. We wholeheartedly invite our fellow scholars to explore this important, overlooked, yet opportunities-filled realm.

2. Related Work

Model editing is a field of study that rests under the wings of safe and efficient machine learning, since properly edited models are made more reliable without extraordinary costs and severe service interaction like retraining. Given the scarcity of GNN-specific editing methods, we provide a brief walkthrough of model editing development and its taxonomy, and refer our reader to survey works such as Mazzia et al. (2023), Zhang et al. (2024b), and curated lists like *KnowledgeEditingPapers* for a more up-to-date and comprehensive understanding of model editing (Yao et al., 2023).

Model Editing in Different Domains As mentioned in the previous section, model editing is a popular realm under various domains like Computer Vision (Sotoudeh & Thakur, 2021; Santurkar et al., 2021; Tanno et al., 2022), Natural Language Processing (Meng et al., 2023b;a; Zhu et al., 2020; Cao et al., 2021; Mitchell et al., 2022), interdisciplinary areas between the two such as Multimodal Models (Cheng et al., 2023), or some more specific areas like Safety-critical Systems (Yu et al., 2023). Further, model-agnostic editing techniques such as ENN by Sinitsin et al. (2020) (which also happen to be the first editing work, to the best of our knowledge) or even general fine-tuning techniques such as Adapter-tuning (Houlsby et al., 2019) and LoRA (Hu et al., 2022) may serve as a potential vehicle for the editing task.

Model Editing for Graph Neural Networks To the best of our knowledge, EGNN by Liu et al. (2023a) is the only GNN-specific model editing technique prior to our work. Though some general, model-agnostic methods like ENN (Sinitsin et al., 2020) and fine-tuning-based techniques like Adapter-tuning (Houlsby et al., 2019) are technically GNN-compatible, they cannot produce even grossly acceptable performance often due to having unaligned assumptions of (graph) data or lack of accommodation of the neighborhood aggregation mechanism, which happens to be two major ingredients to GNNs. This is evidenced by simply fixing the output of one instance (a.k.a. single-edit) with ENN or Adapter-tuning will cause more than 92.4% of general accuracy drop, making the edited model completely unusable (Table 6).

While EGNN (Liu et al., 2023a) does provide a massive performance improvement upon previous migrated approaches in terms of the single-edit evaluation, it still severely suffers from its low Sequential Edit Robustness. An EGNN-edited GNN will have up to 49.3% accuracy drop upon sequentially editing more than ten examples (Table 5). Considering there are almost always more than ten high-profile errors to take care of during the span of a model's life, EGNN, unfortunately, cannot fulfill the editing duty under a practical context.

However, we authors believe EGNN's main contribution lies in its analytical part rather than its solutions. EGNN is the first work to discover that the GNN architecture is naturally more sensitive to editing influence in comparison to other common architectures (e.g., MLP). More on EGNN's contribution in Section 4.1.

3. Preliminary: Formalizing Model Editing for Graph Neural Networks

Given that the task of general model editing was first proposed by Sinitsin et al. (2020) around 4 years ago, many of its terms and objectives still lack consistency amongst different scholars and literature (Mazzia et al., 2023). Added with the extreme scarcity of GNN editing works, the general framework of model editing might be burdensome for readers in the graph learning community to grasp, let alone taking graph-specific settings into consideration. Thus, we hereby formally conceptualize the GNN (sequential) editing task paradigm, provide practical guidance regarding data management, and define its evaluation criteria.

3.1. The Task Paradigm

Suppose a GNN model M_0 trained with training dataset D_{train} is then evaluated on the test set D_{test}. Let us assume that the evaluation result of output $M_0(D_{\text{test}})$ indicates the model is generating undesired outputs on $e_1 = (X_{e_1}, Y_{e_1})$ for $e_1 \in D_{\text{test}}$, where (X_{e_1}, Y_{e_1}) are the features and label of e_1, such that $M_0(e_1) \neq Y_{e_1}$. If this e_1 is considered "edit worthy" due to the potential high-profile impact it entails,

e_1 is considered the *editing target* at hand.

We would then like to perform the editing operation `Edit` upon M and editing target e_1, resulting in an edited model $\texttt{Edit}(M_0, e_1) = M_1$, where a successful edit means $M_1(e_1) = Y_{e_1}$, correcting the previously wrong output of e_1 while maintaining similar performance in regards to inputs that are unrelated to e_1 — i.e., we'd like to $M_0(D_{\text{test}} \backslash e_1) \approx M_1(D_{\text{test}} \backslash e_1)$.

Now suppose this edited model M_1 is producing another high profile error e_2 for $e_2 \in D_{\text{test}}$. We should ideally have $M_2 = \texttt{Edit}(M_1, e_2)$ where $M_2(e_1, e_2) = (Y_{e_1}, Y_{e_2})$ and $M_1(D_{\text{test}} \backslash \{e_1, e_2\}) \approx M_2(D_{\text{test}} \backslash \{e_1, e_2\})$. Given that editing targets e_1 and e_2 likely only take a negligible portion of total dataset D_{test} (as otherwise the model would be considered improperly trained, which is a separate issue) we may relax this objective to $M_1(D_{\text{test}}) \approx M_2(D_{\text{test}})$ and formalize the following task objective for model editing:

$$\begin{gathered}\text{Suppose } M_{n-1}(e_n) \neq Y_{e_n}. \\ \text{We would like to have } M_n = \texttt{Edit}(M_{n-1}, e_n), \\ \text{where } M_n(e_1, \ldots, e_n) = (Y_{e_1}, \ldots, Y_{e_n}) \\ \text{and } M_1(D_{\text{test}}) \approx M_2(D_{\text{test}}) \approx \cdots \approx M_n(D_{\text{test}}),\end{gathered}$$

where n denotes the n-th number of editing targets.

For this paper, we limit our task to node classification where the editing target e is always a node. This decision is made upon the observation that the majority of high-stake GNN applications fall under the realm of node classification tasks and, more importantly, given that GNN model editing is a task that is foreign to the graph learning community, we would like our paper to be lightweight to deliver more insights and guidance on how to approach the graph editing problem. We encourage our fellow scholars to investigate the editing potential of other graph learning tasks.

3.2. The Evaluation Criteria

There are mainly two (series of) evaluation criteria under the model editing paradigm, which are straightforward to understand from a glance:

- **Success Rate (SR):** Indicating an `Edit` operation is successful. For a single-edit, suppose we'd like to conduct $\texttt{Edit}(M_{i-1}, e_i) = M_i$, if $M_i(e_i) = Y_{e_i}$ then the *Success Rate* is 100%. Otherwise, it is 0%. This is considered the paramount metric for model editing, as we'd like to deliver a guaranteed patch to an editing-worthy error.
- **Test Drawdown (DD):** Indicating the gap between pre-edit accuracy and post-edit accuracy. Again, suppose we have $\texttt{Edit}(M_{i-1}, e_i) = M_i$, the *Test Drawdown* is $Acc(M_0(D_{\text{test}})) - Acc(M_i(D_{\text{test}}))$ for $Acc()$ being the accuracy of the given output. Thus, the smaller the drawdown, the better the performance. Yielding negative drawdown after a successful edit means not only the edited model patched the error case, but also improved the model's generalizability.

However, such criteria can get complicated under a sequential context. E.g., under a chain of successful editing events like $\texttt{Edit}(M_{i-1}, e_i) = M_i \rightarrow \texttt{Edit}(M_i, e_{i+1}) = M_{i+1} \ldots$, it is possible to discover $M_{i+1}(e_i) \neq Y_{e_i}$; meaning the later `Edit` operation has overwritten a certain previous edit, causing the current edited model M_{i+1} to produce wrong output on a supposedly already patched instance (in this case, e_i). Suppose the model owner only cares about the Success Rate at the time of a particular `Edit` with regards to a specific editing target (e.g., e_{i+1}), then, in this case, the owner shall have $\text{SR} = 100\%$. However, this is certainly an unfaithful reflection of the reliability of the model, as it cannot correctly process previously edited targets like e_i.

To provide a faithful evaluation of the model, we utilize **i-th Edits Test Drawdown** and **i-th Edits Success Rate**. The former is simply the Test Drawdown of a model's accuracy after experiencing the i-th `Edit` operations, yet the latter is defined as the accuracy of all i editing targets. Thus, a 100% i-th Edits Success Rate implies the edited model M_i is able to provide the correct output for all editing targets $\{e_1, e_2, \ldots, e_i\}$ it has exposed to at the time of the i-th edits, forming a reliable metric to monitor the model's performance under a sequential editing task.

Similar to the challenge faced by Success Rate, we note that metrics like the i-th Edits Test Drawdown can only partially reflect a model's performance under a sequential editing task. Suppose one is only monitoring the 10-th, 20-th, and 30-th Test Drawdown, it will not realize the model might have a drastic peak of Test Drawdown after, for example, the 16-th Edits. To account for such possibilities, we further monitor the model's **Max Drawdown** and **Average Drawdown** during the lifespan of the sequential editing task. Following the same spirit, we also report **Average Success Rate** to provide a single scalar value that is reflective of a model's patching effect under a sequence of editing operations.

Due to page limitation, we refer our readers to Appendix A.1 for data management guidelines regarding sequential graph editing, where we discuss what would be the recommended practice to obtain editing targets, as well as test samples, under a graph-specific context.

4. Motivation

4.1. Prior Art's Take: Neighborhood Aggregation Makes GNNs Intrinsically Editing-Sensitive

Editable Graph Neural Network (EGNN) by Liu et al. (2023a) is, to the best of our knowledge, the only GNN-

specific model editing method available prior to our work. As mentioned in our Related Works section (§2), EGNN provides massive gains on the single-edit metric compared to previous GNN-compatible editing methods, with our replication below confirming this effect (Table 1).

Aside from its performance improvement, we authors believe the main contribution of EGNN is that it reveals that GNN as an architecture is naturally unrobust to edits due to its neighborhood aggregation mechanism, in contrast to topology-free network architectures like multi-layer perceptron (MLP). Here, we verify MLP's vastly superior editing robustness over various popular GNN backbones with a simple toy experiment (Table 1).

Table 1. Different Architectures' Robustness Against a Single Edit. "PE Acc." = Pre-edit Acc., "DD" = (Test) Drawdown (lower is better), and "SR" = Success Rate (higher is better). Please refer to Section 3.2 for details regarding such metrics.

Dataset	Backbone	PE Acc.	DD	SR
ogbn-arxiv	**GCN**	**0.703**	60.5	1.0
	GraphSAGE	0.685	54.4	1.0
	GIN	0.662	61.9	1.0
	MLP	0.526	**1.9**	1.0

We observe that in this vanilla setting (standard gradient descent towards one editing target until its output is corrected), GNN variants are indeed much more editing-sensitive in comparison to MLP.

EGNN's Technical Approach. Following these observations, the authors of EGNN (Liu et al., 2023a) argued that the best way to resolve the editing-sensitive nature of GNNs is to completely side-step it. In particular, **EGNN proposes a frozen GNN + active MLP Ensemble** with the weights of the GNN frozen after the initial training period (serving as M_0 according to languages of the Preliminary section (§3)). The active MLP subsequently updates its weights to realize the editing effects, leveraging their naturally robust-to-edit character observed above (Table 1).

The parallel design of EGNN is particularly clever as it additionally addresses efficiency concerns, which is often a challenge for fine-tuning-based model editing methods, where a large memory footprint is required.

4.2. Prior Art's Struggles: Sequential Edits

Despite EGNN's intuitive design, it cannot handle the scrutiny of sequential edits. This is evidenced by Table 2 below, where EGNN-edited GNNs face an at most 22.6% test drawdown when used for 50 sequential edits. More severely, the average success rate of EGNN across 50 edits is at most 72.1% and as low as 37.5%, suggesting massive network overwriting issues among different editing targets as described in Section 3.2.

Table 2. EGNN (Liu et al., 2023a) under Sequential Edits.

Backbone (PE Acc.)	Test Drawdown (DD) 1th	10th	25th	50th	Avg DD	Avg SR
ogbn-products						
GCN (74.90%)	0.6	38.0	30.8	34.1	30.4	0.721
Graph-SAGE (76.37%)	0.1	3.7	16.6	16.9	10.3	0.654
Amazon Computers						
GCN (85.77%)	1.8	11.4	39.8	27.2	22.6	0.465
Graph-SAGE (83.23%)	2.2	49.0	21.2	6.7	16.0	0.375

We believe it is fair to conclude, from the aforementioned results, that **EGNN is not usable on a practical scale due to its lack of sequential edit robustness**. Under sequential environments, the EGNN-induced Test Drawdown is far too large to keep the edited model within usable margins. Yet, the low Success Rates also indicate an EGNN-edited model cannot provide a guaranteed patch under a sequential context, with the supposedly edited model still producing undesired outputs on previously discovered failure cases. Given model editing techniques are often used to address high-profile failure cases in an urgent manner, a low editing success rate would be a major disqualifying factor.

We further note that according to EGNN's own report (Table 2), other GNN-compatible editing methods are vastly worse than EGNN, meaning that there currently exists no model editing method usable for GNN maintenance in a practical context.

4.3. Are Neighborhood Aggregation and Model Editing Absolutely Incompatible? — Not Really.

Although EGNN makes the valid and natural case that neighborhood aggregation causes GNNs to be intrinsically less robust than architectures like MLP, **whether or not there exists a way to make a vanilla GNN model perform better under editing tasks remains largely unexplored.** This is because EGNN simply side-steps the node aggregation issue they discovered with their frozen GNN + active MLP ensemble design, where the MLP is entirely parallel to the GNN.

To fill the gap, we inspect EGNN's procedure alongside its undesired sequential edit results, where we notice EGNN is essentially doing gradient descent towards *just one particular edit target* during each of its `Edit` operations. **We suspect this might be a recipe for overfitting**, as even the largest node feature we tested only has a dimension of `6805`, making it an easy victim to be overfitted by a GNN model, or a two-to-four layer MLP utilized in EGNN.

To confirm our hypothesis, we introduce regularization effects to the editing procedure by simply mixing in 100 randomly selected training samples from training set D_{train} (denoted as ts^{100} for $ts^{100} \in D_{\text{train}}$) to be batched together with the editing target at hand. Namely, instead of the original $\texttt{Edit}(M_{i-1}, e_i) = M_i$ operation, where model M_{i-1} is tasked to conduct gradient descent until it correctly classify e_i, we will now have $\texttt{Edit}(M_{i-1}, \{e_i, ts^{100}\}) = M_i$, where we will conduct gradient descent upon the batch of e_i and ts^{100} until the batch's output is correct.

Table 3. Batched vs. Non-Batched Performance under Sequential Edits. "DD" of "{10, 25, 50}-th" refers to the Test Drawdown after {10, 25, 50}-th edits.

Backbone (PE Acc.)	**Method** (Batched?)	**Test Drawdown (DD)** 10th	25th	50th	**Avg DD**
Amazon Computers					
GCN (85.77%)	GNN (✗)	32.9	67.1	82.2	63.9
	GNN (✓)	8.0	4.4	4.3	4.7
	Frozen GNN+MLP (✗)	11.4	39.8	27.2	22.6
	Frozen GNN+MLP (✓)	**-0.6**	**-0.6**	**-0.6**	**-0.6**
ogbn-arxiv					
Graph-SAGE (68.45%)	GNN (✗)	64.9	65.2	64.7	59.3
	GNN (✓)	7.2	9.3	5.2	7.8
	Frozen GNN+MLP (✗)	10.0	17.6	15.0	17.8
	Frozen GNN+MLP (✓)	**1.4**	**1.4**	**1.4**	**1.4**

We observe that batching random training samples with the edit target dramatically improves model performance on DD-related metrics. We highlight that this improvement is present — and especially massive — upon vanilla GNN architectures (without the parallel MLP introduced by EGNN). This suggests while it is true that neighborhood aggregation makes GNN intrinsically editing-sensitive, there are still ways to mitigate this inherent disadvantage. We further note that overfitting reduction via batched gradient descent is merely one such channel, and we encourage our fellow scholars from the graph-learning community to explore more refined designs.

4.4. Takeaway: Overfitting Mitigation Helps (Sequential) Editing Robustness

Through the investigations conducted in this section (§4), we may conclude that there is a strong connection between overfitting and editing robustness. This conclusion — although never been made under a graph editing context — is non-surprising by nature, as a sequence of individual editing targets with low feature dimensions is the perfect victim for overfitting. Yet, the large Test Drawdown reading we observed is more or less just another angle of the *catastrophic forgetting* (Sinitsin et al., 2020; Mazzia et al., 2023; Zhang et al., 2024a; Cheng et al., 2023; Houlsby et al., 2019; Zhu et al., 2020; Cao et al., 2021; Mitchell et al., 2022), a phenomenon that scholars have observed and studied under various different contexts. Specifically, using batching to reduce overfitting during editing operations has been previously explored in literature like MEND (Mitchell et al., 2022).

Since we now know the tie between (sequential) editing robustness and overfitting, we may afford to integrate some overfitting mitigation techniques into the graph editing procedure, aiming for improved editing performance. More on this in the Proposed Method section below (§5).

5. Proposed Method

Inspired by the connection between overfitting and sequential editing robustness discussed in Section 4, we propose Sequentially Editable Graph Neural Networks, or SEED-GNN. SEED-GNN employs several lightweight, architecture-agnostic batching operations to provide edit-aware benefits such as overfitting mitigation and overwriting prevention, while simultaneously being seamlessly integrable into the frozen GNN + active MLP ensemble pipeline proposed in EGNN (Liu et al., 2023a) to leverage its efficiency advantages.

5.1. Following the Frozen GNN + Active MLP Architecture for Efficient Editing

It is intuitive that an effective model editing method should be able to patch a sequence of errors with consistently high success rates (in regard to all editing targets throughout the lifespan of the model) while inducing as low of a test drawdown as possible. However, another important aspect — which often constitutes a qualify or disqualify factor under a practical context — is **the efficiency of the editing method**. Suppose the cost of a certain editing method exceeds the cost of retraining; while being and extreme example unlikely to happen in practice, there will be little to no reason to conduct model editing under this scenario. On the other hand, the patching of high-profile failure cases is often time-sensitive, where a runtime and resource-efficient model editing method is often preferred.

EGNN by Liu et al. (2023a) provides decent efficiency benefits over vanilla finetuning-based methods (such as full model finetune). By only updating the stitched MLP weights while keeping the GNN part frozen, EGNN avoids the expenses that are particularly time/memory-intensive for large-scale graph datasets, as one can "cache" the output from frozen GNN after one pass of the input to avoid future `forward()` needs, while there is no need to store the gradients and optimizer states corresponding to the GNN weights; as it is frozen to begin with. As a method aiming for practical usability, we follow the EGNN ensemble design for its efficiency benefits. Additionally, having an aligned architecture may also help us to tribute the performance gain to the following strategies we proposed, but not

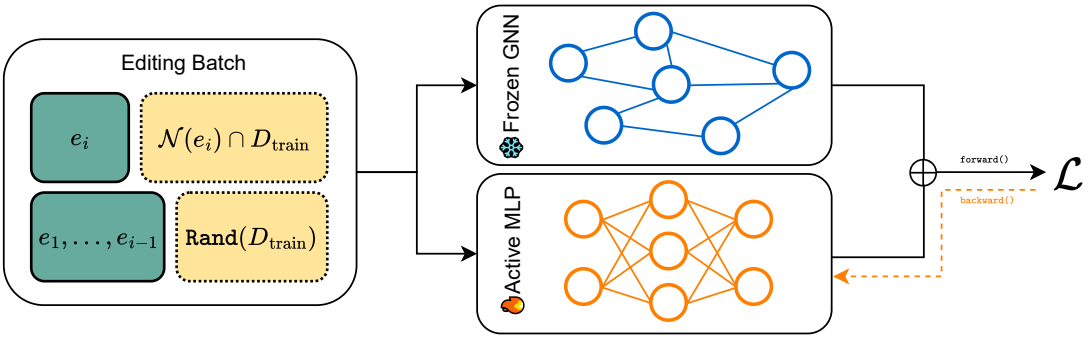

Figure 1. **Main Procedure of SEED-GNN.** To edit an editing target e_i, we form an editing batch consisting of four types of components: the current editing target e_i itself, previous editing targets $e_1, \ldots, e_{i-1}$, e_i's neighbors that happen to be in the train set $\mathcal{N}(e_i) \cap D_{\text{train}}$, and randomly selected training samples from the train set $\texttt{Rand}(D_{\text{train}})$. We follow the Frozen GNN + Active MLP design proposed in EGNN (Liu et al., 2023a), where we combine the output of the GNN and the MLP part as the final output of SEED-GNN's `forward()`, but only `backward()` update the MLP weights to host the editing effect. The weight update terminates either because the predictions of e_i and $e_1, \ldots, e_{i-1}$ are corrected, or if the *Steps* budget in Table 7 is fully spent. Please refer to Section 5 for details.

some inherent properties of a different network architecture. We refer our reader to Appendix C for runtime and memory-related information.

5.2. Edit-aware Training Sample Mix-up Addresses Neighborhood Corrosion

Following the discovery that batch editing may help with editing robustness (§4.3), we further investigate what kind of data is considered beneficial when batched together with editing targets. Recall that EGNN (Liu et al., 2023a) finds that node aggregation is detrimental to editing robustness because of the error propagation to its neighbors. Here, we investigate the k-hop accuracy of a randomly selected misclassified node (a.k.a. an editing target) and find that **most of the misclassified node's k-hop neighbors are, in fact, correctly classified by the pre-edit GNN model in the first place**. However, a successful (but vanilla) patch of this editing target would corrode its neighbors, resulting in a massive accuracy drop among its k-hop neighbors (Table 4) — we denote this phenomenon as editing-induced neighborhood corrosion.

Table 4. Pre-edit vs. Post-edit k-Hop Acc.

	GCN	Cora	Amazon Computers	Amazon Photo	Coauthor-CS
1-hop	Pre-edit Acc.	73.4	54.4	73.2	85.2
	Post-edit Acc.	54.8	31.3	27.6	47.8
2-hop	Pre-edit Acc.	77.9	71.9	80.9	91.7
	Post-edit Acc.	64.6	19.9	27	61.9
3-hop	Pre-edit Acc.	80.3	79.7	82.7	92.2
	Post-edit Acc.	72.9	21.2	27.8	72.9

This observation is another angle of the same overfitting issue discussed in Section 4.3, and it is likely baked in the GNN aggregation mechanism. However, known that 1) batching editing targets with randomly selected training samples helps editing robustness (Table 3), and 2) most of an editing target's neighbors are correct before editing (Table 4). We can leverage them together by forcibly introducing the neighbors of the editing target to be batched with the editing target[1], so that these neighbors can receive some protection for directly contributing to the (batched) gradient calculation during editing. Formally speaking, following the above strategy, suppose model M_{i-1} is facing an edit target e_i. We shall have:

$$\texttt{Edit}(M_{i-1}, \{\, e_i,\ \mathcal{N}(e_i) \cap D_{\text{train}}, \texttt{Rand}(D_{\text{train}})\}) = M_i$$
$$\text{where } |\mathcal{N}(e_i) \cap D_{\text{train}}| = \alpha \cdot \beta$$
$$\text{and } |\texttt{Rand}(D_{\text{train}})| = (1-\alpha) \cdot \beta,$$

in such case, $\{\, e_i,\ \mathcal{N}(e_i) \cap D_{\text{train}}, \texttt{Rand}(D_{\text{train}})\}$ is the editing batch, where $\mathcal{N}(e_i)$ is the neighbors (regardless of how many hops) of e_i, β is the number of extra samples we would like to include in the batch outside the editing targets, and α is a balancing factor to control the ratio between including more eligible neighbors of e_i, or more randomly selected training samples. Suppose one would like to have 100 extra samples included in the batch, with 25% of them being eligible neighborhood nodes of the editing target and 75% of them randomly selected from the training set; we'd have $\alpha = 0.25$ and $\beta = 100$ — which is also the setting for all reported experiments of SEED-EGNN. We share more details of this mix-up in Algorithm 1 and provide ablation studies regarding α and β in Appendix B.

5.3. Edit-aware Incremental Batching and Stoppage

One significant issue of the above method (or its non-batched variants) is the model will only face just one editing target at a time, resulting in catastrophic forgetting where the latter editing targets overwrite the patch of previously

[1] Of course, to prevent data leakage, only neighbors who belong to the training set are eligible. Please refer to Algorithm 1 for details.

concluded edits. This is evidenced by the low success rate visible in Table 5 and Table 6. To address this issue, we can simply include all historical editing targets in the editing batch, making the editing batch essentially $\{ e_1, e_2, \ldots, e_i, \mathcal{N}(e_i) \cap D_{\text{train}}, \texttt{Rand}(D_{\text{train}})\}$ during the i-th edit. Where an `Edit` operation will perform gradient descent with this editing batch as input until all editing targets $\{e_1, e_2, \ldots, e_i\}$ are patched, or a certain iteration budget is expensed.

We have conducted ablation studies to confirm the cumulative effectiveness of our proposed procedure described above. We invite interested readers to Appendix B for a deeper and quantitative dive into the dynamics between those sub-strategies. We also provide ablation studies regarding the two method-specific hyperparameters (α and β) and design (incremental batching) of SEED-GNN, again at Appendix B.

6. Experiments and Results

6.1. Experiment Setups

Models, Datasets, and Compared Methods. To assess the effectiveness of SEED-GNN, we selected seven benchmark datasets from various fields, including common small-scale and large-scale graph datasets like Cora (McCallum et al., 2000), Amazon Photo (Shchur et al., 2018), Coauthor-CS (Shchur et al., 2018), Amazon Computers (Shchur et al., 2018), ogbn-arxiv, and ogbn-products (Hu et al., 2020). Since model editing is aimed to address high-profile failure cases, we additionally provide results on YelpCHI (Rayana & Akoglu, 2015), a real-word collected fraud detection dataset. Model coverage-wise, we integrated SEED-GNN with four established GNN architectures, namely, GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2017), and GIN (Xu et al., 2019). For clarity, the evaluation of SEED-GNN is conducted in an inductive setting whenever possible, meaning the model is trained on a subset of the graph that includes only training nodes and then tested on the entire graph to gauge its effectiveness.

For methods, we compare our SEED-GNN with all visible editors under the GNN editing literature, namely, direct finetuning (FT), ENN by Sinitsin et al. (2020), and EGNN by Liu et al. (2023a). We additionally provide comparisons with two common parameter-efficient finetuning (PEFT) methods: Adapter-tuning (Houlsby et al., 2019) and LoRA (Hu et al., 2022), due to their intuitive compatibility with model editing tasks. We follow EGNN's (Liu et al., 2023a) setting and pipeline for an aligned comparison, unless otherwise specified in Appendix B.

Evaluation Criteria We evaluate the effectiveness of different methods by i-th Edits **Test DrawDown (DD)** — which is the difference between pre-edit accuracy and after performing the i-th edit accuracy, where a smaller drawdown indicates a better editor locality — and i-th Edits **Success Rate (SR)**, which is defined as the success rate of patching all i of editing targets. We adopted these criteria in a sequential setting, and we strongly encourage our readers to inspect Section 3.2 for their detailed definitions.

Table 5. GNN Model Editing Experiments on Large Scale Graphs. Please refer to Table 6 and §3.2 for further metrics specifications.

Backbone (PE Acc.)	Method	Test Drawdown (Success Rate) 1th	10th	25th	50th	Max DD	Avg DD	Avg SR
Amazon Computers	13,381 Nodes	245,778 Edges	10 Classes	767 Features				
GCN (85.77%)	FT	77.1 (1.0)	32.9 (0.4)	67.1 (0.12)	82.2 (0.28)	85.1	63.9	0.30
	ENN	2.2 (1.0)	77.3 (0.3)	77.3 (0.16)	77.3 (0.1)	77.3	75.7	0.21
	EGNN	1.8 (1.0)	11.4 (0.7)	39.8 (0.48)	27.2 (0.38)	64.0	22.6	0.47
	Adapter	77.3 (1.0)	77.3 (0.3)	82.3 (0.2)	82.9 (0.12)	85.7	73.2	0.24
	LoRA	2.3 (0.0)	2.3 (0.2)	2.3 (0.28)	1.5 (0.12)	2.3	2.1	0.19
	SEED-GNN	**0.6 (1.0)**	**-0.9 (1.0)**	**-0.1 (1.0)**	**2.7 (1.0)**	**3.3**	**1.1**	**0.99**
Graph-SAGE (83.23%)	FT	25.3 (1.0)	76.2 (0.3)	66.5 (0.28)	69.3 (0.14)	82.5	60.8	0.31
	ENN	**-0.6 (1.0)**	74.7 (0.1)	74.7 (0.12)	74.7 (0.16)	74.7	73.0	0.15
	EGNN	2.2 (1.0)	49.0 (0.5)	21.2 (0.32)	6.7 (0.28)	55.3	16.0	0.38
	Adapter	82.2 (1.0)	70.6 (0.3)	70.5 (0.52)	66.7 (0.16)	83.1	64.4	0.34
	SEED-GNN	1.9 (1.0)	**0.5 (1.0)**	**-0.9 (1.0)**	**-0.2 (1.0)**	**3.5**	**0.0**	**1.0**
GIN (66.11%)	FT	49.7 (1.0)	48.3 (0.3)	49.4 (0.48)	50.4 (0.48)	66.0	51.0	0.33
	ENN	57.7 (0.0)	65.5 (0.2)	65.8 (0.2)	36.7 (0.14)	66.0	46.6	0.15
	EGNN	**5.7 (1.0)**	0.3 (0.5)	6.9 (0.4)	-6.5 (0.46)	19.3	3.40	0.51
	Adapter	70.9 (1.0)	86.6 (0.1)	86.4 (0.2)	87.3 (0.1)	87.3	73.5	0.20
	SEED-GNN	-3.1 (0.0)	**-4.1 (1.0)**	**-9.6 (1.0)**	**-10.2 (1.0)**	**3.1**	**-8.0**	**0.95**
ogbn-arxiv	169,343 Nodes	1,166,243 Edges	40 Classes	128 Features				
GCN (70.26%)	FT	60.5 (1.0)	42.3 (0.5)	67.5 (0.12)	66.5 (0.12)	69.1	58.5	0.22
	ENN	48.2 (0.0)	48.2 (0.0)	48.2 (0.0)	48.2 (0.02)	48.2	48.2	0.01
	EGNN	3.0 (1.0)	9.2 (0.9)	17.9 (0.2)	4.2 (0.14)	58.4	13.0	0.40
	Adapter	48.7 (1.0)	70.0 (0.1)	64.3 (0.24)	68.4 (0.02)	70.0	60.2	0.23
	LoRA	3.9 (0.0)	3.9 (0.1)	3.9 (0.04)	3.9 (0.1)	3.9	3.9	0.08
	SEED-GNN	**-3.7 (1.0)**	**4.7 (1.0)**	**5.3 (1.0)**	**6.2 (1.0)**	**9.8**	**6.1**	**1.0**
Graph-SAGE (68.45%)	FT	54.4 (1.0)	64.9 (0.1)	65.2 (0.12)	64.7 (0.1)	67.7	59.3	0.22
	ENN	66.2 (0.0)	68.0 (0.0)	68.0 (0.04)	68.0 (0.06)	68.0	68.0	0.04
	EGNN	0.6 (1.0)	10.0 (0.5)	17.6 (0.32)	15.0 (0.36)	39.8	17.8	0.40
	Adapter	67.2 (1.0)	58.9 (0.2)	46.9 (0.32)	64.6 (0.14)	68.3	58.9	0.17
	LoRA	2.7 (0.0)	2.7 (0.0)	2.9 (0.2)	2.1 (0.1)	3.7	2.3	0.11
	SEED-GNN	**0.1 (1.0)**	**5.8 (1.0)**	**5.5 (1.0)**	**4.8 (1.0)**	**11.0**	**4.9**	**1.0**
GIN (66.17%)	FT	61.9 (1.0)	46.0 (0.2)	63.6 (0.16)	63.7 (0.02)	65.9	59.9	0.16
	ENN	64.9 (0.0)	41.8 (0.1)	44.9 (0.16)	44.4 (0.14)	66.1	51.6	0.10
	EGNN	**0.2 (1.0)**	34.6 (0.6)	22.4 (0.32)	30.0 (0.22)	53.0	17.0	0.35
	Adapter	65.3 (1.0)	57.9 (0.2)	63.6 (0.12)	60.1 (0.12)	65.3	54.6	0.229
	SEED-GNN	0.7 (1.0)	**1.1 (0.8)**	**4.9 (0.96)**	**6.7 (0.98)**	**8.6**	**5.2**	**0.945**
ogbn-products	2,449,029 Nodes	61,859,140 Edges	47 Classes	218 Features				
GCN (74.90%)	FT	4.1 (1.0)	22.9 (0.7)	62.8 (0.48)	64.4 (0.6)	67.5	42.69	0.67
	ENN	46.6 (0.0)	70.4 (0.0)	70.4 (0.0)	70.4 (0.04)	70.4	69.9	0.02
	EGNN	**0.6 (1.0)**	38.0 (0.7)	30.8 (0.76)	34.1 (0.52)	40.6	30.4	0.72
	Adapter	68.8 (1.0)	70.3 (0.1)	68.1 (0.16)	70.1 (0.1)	73.6	63.3	0.21
	LoRA	4.4 (1.0)	3.9 (0.2)	1.8 (0.32)	1.7 (0.34)	4.4	2.7	0.30
	SEED-GNN	3.4 (1.0)	**7.7 (1.0)**	**7.0 (1.0)**	**10.7 (1.0)**	**12.4**	**8.3**	**1.0**
Graph-SAGE (76.37%)	FT	13.3 (1.0)	56.0 (1.0)	59.6 (0.88)	60.7 (0.6)	66.4	55.9	0.72
	ENN	2.3 (1.0)	53.7 (0.1)	72.1 (0.04)	72.1 (0.08)	72.1	62.9	0.11
	EGNN	**0.1 (1.0)**	3.7 (0.9)	16.6 (0.36)	16.9 (0.52)	23.3	10.29	0.65
	Adapter	49.8 (1.0)	68.8 (0.1)	71.0 (0.04)	74.0 (0.04)	75.2	64.2	0.23
	LoRA	1.3 (0.0)	1.3 (0.1)	1.4 (0.12)	1.6 (0.14)	1.6	1.4	0.121
	SEED-GNN	1.8 (1.0)	**2.5 (1.0)**	**3.2 (1.0)**	**6.2 (1.0)**	**6.6**	**3.6**	**1.0**
GIN (65.79%)	FT	64.5 (1.0)	61.2 (0.2)	61.1 (0.12)	63.8 (0.12)	64.5	57.99	0.22
	EGNN	**-0.2 (1.0)**	22.0 (0.8)	16.0 (0.56)	15.4 (0.58)	24.7	18.2	0.62
	Adapter	38.8 (1.0)	61.4 (0.1)	63.1 (0.04)	58.9 (0.06)	65.7	54.1	0.23
	SEED-GNN	-0.3 (0.0)	**1.4 (0.9)**	**3.5 (0.96)**	**1.8 (1.0)**	**4.3**	**2.3**	**0.93**

6.2. Results and Conclusion

Our proposed method, SEED-GNN, has showcased significant advantages in almost all reported entries. We kindly direct our readers' attention to two reflective metrics: Average Success Rate (Avg SR) and Average Test Drawdown (Avg DD) (Table 5 and 6). It is clear that no other editing method can maintain a ≈ 1.0 Avg SR except SEED-GNN, yet SEED-GNN may deliver a constant $\leq 10\%$ Avg DD under all reported settings — a result vastly below all comparing approaches. Additionally, we provide SEED-GNN's results against a few alternative approaches that take advan-

Table 6. GNN Model Editing Experiments on Small Scale Graphs. "DD" = Test Drawdown (lower is better), 'SR" = Success Rate (higher is better, where $\neq 1.0$ implies insecure edits), "PE Acc." = Pre-edit Acc. Please refer to Section 3.2 for further metrics specifications.

Backbone (PE Acc.)	**Method**	**Test Drawdown (Success Rate)** 1th	10th	25th	50th	**Max DD**	**Avg DD**	**Avg SR**
Cora 2,485 Nodes 5,069 Edges 7 Classes 1,433 Features								
GCN (89.80%)	FT	4.4 (1.0)	28.2 (0.7)	17.0 (0.76)	37.0 (0.62)	43.2	21.1	0.73
	ENN	1.2 (1.0)	13.6 (1.0)	17.8 (0.96)	43.4 (0.98)	46.8	21.6	0.87
	EGNN	0.6 (1.0)	13.6 (0.8)	14.8 (0.56)	13.2 (0.44)	21.4	10.6	0.60
	Adapter	57.8 (1.0)	76.0 (0.3)	82.2 (0.08)	76.0 (0.26)	82.4	74.5	0.231
	SEED-GNN	**0.6 (1.0)**	**2.4 (1.0)**	**3.2 (1.0)**	**2.6 (1.0)**	**5.4**	**3.2**	**1.0**
Graph-SAGE (86.60%)	FT	1.8 (1.0)	23.0 (0.9)	27.2 (1.0)	41.4 (0.68)	47.2	28.2	0.85
	ENN	**-1.2 (1.0)**	13.6 (0.9)	28.4 (0.68)	36.0 (0.66)	39.0	25.6	0.75
	EGNN	1.8 (1.0)	27.2 (0.9)	5.6 (0.52)	16.0 (0.44)	28.8	10.8	0.62
	Adapter	74.6 (1.0)	75.2 (0.1)	70.8 (0.36)	79.0 (0.12)	79.2	74.3	0.251
	SEED-GNN	-0.4 (1.0)	**-0.8 (1.0)**	**0.2 (1.0)**	**1.0 (1.0)**	**3.0**	**1.0**	**1.0**
GAT (87.6%)	FT	8.6 (1.0)	8.0 (0.9)	33.2 (0.88)	44.2 (0.68)	42.2	24.2	0.75
	ENN	1.6 (1.0)	11.2 (0.8)	12.4 (0.64)	16.8 (0.54)	19.8	13.2	0.69
	EGNN	0.4 (0.1)	10.4 (0.9)	2.0 (0.36)	7.6 (0.46)	12.2	5.2	0.55
	Adapter	55.6 (1.0)	80.0 (0.1)	73.8 (0.16)	55.6 (0.24)	80.0	70.40	0.24
	SEED-GNN	**0.4 (1.0)**	**1.0 (1.0)**	**1.6 (1.0)**	**1.8 (1.0)**	**3.2**	**1.2**	**1.0**
GIN (84.2%)	FT	30.2 (1.0)	76.4 (0.1)	70.2 (0.04)	76.8 (0.08)	76.8	65.1	0.27
	ENN	45.4 (0.0)	35.2 (0.3)	49.4 (0.2)	49.8 (0.24)	73.2	49.5	0.24
	EGNN	**0.6 (1.0)**	2.2 (0.5)	11.2 (0.48)	2.6 (0.42)	44.2	10.6	0.46
	Adapter	76.2 (1.0)	52.2 (0.2)	70.4 (0.2)	72.2 (0.22)	76.8	66.0	0.29
	SEED-GNN	-0.4 (0.0)	**0.2 (0.9)**	**1.2 (1.0)**	**1.6 (1.0)**	**2.6**	**0.70**	**0.90**
Amazon Photo 7,487 Nodes 119,043 Edges 8 Classes 745 Features								
GCN (93.81%)	FT	78.0 (1.0)	85.4 (0.5)	62.8 (0.2)	92.6 (0.36)	93.3	80.8	0.25
	ENN	83.7 (0.0)	84.2 (0.0)	84.2 (0.04)	84.2 (0.1)	77.3	75.7	0.21
	EGNN	1.1 (1.0)	36.4 (0.9)	42.5 (0.6)	11.2 (0.38)	45.0	23.4	0.61
	Adapter	93.3 (1.0)	49.7 (0.4)	55.9 (0.12)	88.7 (0.26)	93.3	75.6	0.30
	LoRA	1.0 (1.0)	**1.0 (1.0)**	1.0 (0.12)	0.6 (0.08)	1.0	0.90	0.16
	SEED-GNN	**-0.1 (1.0)**	1.2 (1.0)	**1.1 (1.0)**	**1.1 (1.0)**	**1.8**	**0.8**	**1.0**
Graph-SAGE (94.36%)	FT	60.4 (1.0)	58.1 (0.6)	84.6 (0.28)	93.7 (0.12)	93.7	79.6	0.36
	ENN	66.4 (0.0)	86.9 (0.2)	86.9 (0.24)	86.8 (0.14)	86.9	85.2	0.21
	EGNN	**1.4 (1.0)**	1.0 (0.5)	5.8 (0.36)	10.9 (0.18)	27.9	9.2	0.39
	Adapter	77.4 (1.0)	81.2 (0.7)	85.3 (0.2)	51.2 (0.4)	93.8	75.1	0.31
	SEED-GNN	2.0 (1.0)	**0.8 (1.0)**	**2.3 (1.0)**	**2.3 (1.0)**	**5.4**	**2.2**	**1.0**
GAT (93.15%)	FT	28.4 (1.0)	65.7 (0.6)	54.7 (0.6)	76.4 (0.46)	92.7	53.1	0.65
	ENN	39.1 (0.0)	83.5 (0.1)	83.5 (0.08)	83.5 (0.12)	83.5	81.2	0.11
	EGNN	**-0.6 (1.0)**	17.6 (0.4)	24.3 (0.52)	33.5 (0.42)	82.4	26.0	0.44
	Adapter	88.1 (1.0)	85.7 (0.3)	55.3 (0.16)	83.5 (0.18)	92.6	76.6	0.23
	SEED-GNN	-0.3 (1.0)	**0.5 (1.0)**	**6.0 (1.0)**	**19.9 (0.98)**	**19.9**	**7.3**	**0.99**
GIN (86.11%)	FT	76.8 (1.0)	85.6 (0.1)	85.5 (0.36)	48.2 (0.14)	85.6	72.4	0.22
	ENN	82.5 (1.0)	65.8 (0.3)	76.6 (0.16)	52.1 (0.2)	85.6	64.4	0.27
	EGNN	-1.4 (1.0)	0.0 (0.2)	20.2 (0.44)	4.2 (0.46)	33.1	9.3	0.50
	Adapter	53.6 (1.0)	77.1 (0.2)	58.0 (0.16)	72.7 (0.26)	85.6	72.2	0.21
	SEED-GNN	**-1.7 (1.0)**	**-1.7 (0.9)**	**-4.2 (0.96)**	**-4.9 (1.0)**	**1.7**	**-3.60**	**0.96**
Coauthor-CS 18,333 Nodes 81,894 Edges 15 Classes 6,805 Features								
GCN (94.43%)	FT	5.9 (1.0)	74.6 (0.7)	33.7 (0.8)	59.0 (0.46)	74.6	43.3	0.74
	ENN	0.6 (1.0)	7.6 (0.8)	7.6 (0.72)	6.8 (0.6)	17.3	9.2	0.70
	EGNN	**-0.2 (1.0)**	0.0 (1.0)	1.3 (0.84)	1.7 (0.86)	1.8	0.90	0.91
	Adapter	60.3 (1.0)	94.1 (0.2)	81.4 (0.24)	91.2 (0.16)	94.4	84.3	0.25
	LoRA	0.4 (1.0)	0.4 (0.3)	0.4 (0.2)	0.4 (0.16)	0.4	0.4	0.22
	SEED-GNN	-0.1 (1.0)	**-0.1 (1.0)**	**-0.2 (1.0)**	**-0.1 (1.0)**	**0.1**	**0.0**	**1.0**
Graph-SAGE (95.33%)	FT	4.3 (1.0)	62.1 (1.0)	43.1 (0.84)	66.8 (0.68)	74.4	54.6	0.88
	ENN	7.6 (1.0)	10.2 (0.7)	8.5 (0.56)	18.8 (0.56)	22.0	14.0	0.65
	EGNN	**0.0 (1.0)**	0.0 (0.9)	7.1 (0.8)	3.1 (0.8)	8.2	3.9	0.85
	Adapter	94.9 (1.0)	93.0 (0.3)	93.4 (0.12)	61.2 (0.12)	95.2	86.5	0.19
	SEED-GNN	**0.0 (1.0)**	**-0.1 (1.0)**	**0.2 (1.0)**	**0.7 (1.0)**	**1.5**	**0.5**	**1.0**
GAT (93.78%)	FT	3.2 (1.0)	54.4 (0.7)	57.3 (0.88)	56.2 (0.78)	71.7	50.7	0.86
	ENN	5.0 (1.0)	65.3 (0.3)	62.2 (0.12)	61.8 (0.12)	66.4	60.5	0.22
	EGNN	-0.1 (1.0)	3.4 (1.0)	7.1 (1.0)	10.0 (0.94)	11.0	5.0	0.97
	Adapter	93.5 (1.0)	71.2 (0.3)	86.2 (0.04)	86.9 (0.06)	93.6	84.9	0.16
	SEED-GNN	**-0.3 (1.0)**	**2.2 (1.0)**	**1.4 (1.0)**	**-0.5 (1.0)**	**2.4**	**0.3**	**1.0**
GIN (91.60%)	FT	43.0 (1,0)	84.7 (0.5)	77.3 (0.12)	91.4 (0.08)	91.4	82.2	0.24
	ENN	49.6 (1.0)	91.0 (0.2)	91.0 (0.16)	91.1 (0.08)	91.2	90.0	0.19
	EGNN	-0.2 (1.0)	-0.5 (0.9)	1.8 (0.84)	10.3 (0.74)	11.2	3.3	0.79
	Adapter	78.5 (1.0)	84.7 (0.1)	77.1 (0.12)	91.3 (0.1)	91.6	79.0	0.18
	SEED-GNN	**-0.4 (1.0)**	**-1.0 (1.0)**	**-1.3 (1.0)**	**-1.1 (1.0)**	**-0.4**	**-1.2**	**0.99**

tage of SEED-GNN's batching ingredients in Appendix D.1, as well as evaluating SEED-GNN upon YelpCHI the real-word collected fraud detection dataset in Appendix D.2; both results indicate the superiority of SEED-GNN.

The above observations support our claim that **SEED-GNN is the first and only practically usable GNN model editing method as of today.** And we strongly encourage future study of the graph editing problem, as we wholeheartedly believe the graph-equivalent of fixing misclassifying a street-crossing child in front of a self-driving car of course deserves studying.

Acknowledgments

This research was supported, in part, by NSF Awards OAC-2112606, OAC-2117439, IIS-2224843, and IIS-2310260. This work made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University. We give our special thanks to the CWRU HPC team for their timely and professional help and maintenance. The views and conclusions in this paper are those of the authors and do not represent the views of any funding or supporting agencies.

Impact Statement

Our work advances the field of Graph Neural Network Model Editing, but it also comes with various **constraints and limitations**. Scope-wise, our method is limited to node classification tasks, leaving various other graph learning tasks, such as edge prediction and graph classification, unexplored, though we expect our proposed method to be adaptable to such tasks. Solution-wise, though our method provides massive gains on all criteria compared to the few available prior arts, it is nowhere near perfect — e.g., we purposely highlight the **Max DD** metric in Table 5 and 6, which is often a lot higher than its Avg DD and, certainly is a welcoming aspect for improvement. We caution our readers to directly adopt our method without proper evaluation under high-stake graph learning scenarios, such as the flood prediction scenario mentioned in (Kazadi et al., 2022).

References

Cao, N. D., Aziz, W., and Titov, I. Editing factual knowledge in language models, 2021.

Chang, C.-Y., Chuang, Y.-N., Wang, G., Du, M., and Na, Z. Dispel: Domain generalization via domain-specific liberating. *arXiv preprint arXiv:2307.07181*, 2023.

Chen, T., Zhou, K., Duan, K., Zheng, W., Wang, P., Hu, X., and Wang, Z. Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):2769–2781, 2022.

Cheng, S., Tian, B., Liu, Q., Chen, X., Wang, Y., Chen, H., and Zhang, N. Can we edit multimodal large language models? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13877–13888, December 2023.

Gu, H., Zhou, K., Han, X., Liu, N., Wang, R., and Wang, X. Pokemqa: Programmable knowledge editing for multi-hop question answering. *arXiv preprint arXiv:2312.15194*, 2023.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Hsieh, K., Wang, Y., Chen, L., Zhao, Z., Savitz, S., Jiang, X., Tang, J., and Kim, Y. Drug repurposing for covid-19 using graph neural network and harmonizing multiple evidence. *Scientific reports*, 11(1):23179, 2021.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Kazadi, A. N., Doss-Gollin, J., Sebastian, A., and Silva, A. Flood prediction with graph neural networks. *Climate Change AI. Climate Change AI*, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Li, T., Zhou, Z., Li, S., Sun, C., Yan, R., and Chen, X. The emerging graph neural networks for intelligent fault diagnostics and prognostics: A guideline and a benchmark study. *Mechanical Systems and Signal Processing*, 168: 108653, 2022.

Lina, D. H. and Silva, A. Better fair than sorry: Adversarial missing data imputation for fair gnns, 2024.

Liu, Y. Fairgraph: Automated graph debiasing with gradient matching. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 4135–4139, 2023.

Liu, Z., Zhou, K., Yang, F., Li, L., Chen, R., and Hu, X. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2021.

Liu, Z., Jiang, Z., Zhong, S., Zhou, K., Li, L., Chen, R., Choi, S.-H., and Hu, X. Editable graph neural network for node classifications, 2023a.

Liu, Z., Shengyuan, C., Zhou, K., Zha, D., Huang, X., and Hu, X. Rsc: accelerate graph neural networks training via randomized sparse computations. In *International Conference on Machine Learning*, pp. 21951–21968. PMLR, 2023b.

Mazzia, V., Pedrani, A., Caciolai, A., Rottmann, K., and Bernardi, D. A survey on knowledge editing of neural networks, 2023.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in gpt, 2023a.

Meng, K., Sharma, A. S., Andonian, A., Belinkov, Y., and Bau, D. Mass-editing memory in a transformer, 2023b.

Mitchell, E., Lin, C., Bosselut, A., Finn, C., and Manning, C. D. Fast model editing at scale, 2022.

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

Rayana, S. and Akoglu, L. Collective opinion spam detection: Bridging review networks and metadata. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2015.

Rong, Y., Wang, G., Feng, Q., Liu, N., Liu, Z., Kasneci, E., and Hu, X. Efficient gnn explanation via learning removal-based attribution. *arXiv preprint arXiv:2306.05760*, 2023.

Santurkar, S., Tsipras, D., Elango, M., Bau, D., Torralba, A., and Madry, A. Editing a classifier by rewriting its prediction rules, 2021.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Shu, K., Sliva, A., Wang, S., Tang, J., and Liu, H. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, 19(1):22–36, 2017.

Sinitsin, A., Plokhotnyuk, V., Pyrkin, D., Popov, S., and Babenko, A. Editable neural networks. In *International Conference on Learning Representations*, 2020.

Sotoudeh, M. and Thakur, A. V. Provable repair of deep neural networks, 2021.

Sun, M., Zhou, K., He, X., Wang, Y., and Wang, X. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1717–1727, 2022.

Tanno, R., Pradier, M. F., Nori, A., and Li, Y. Repairing neural networks by leaving the right past behind, 2022.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, C., Lin, Z., Yang, X., Sun, J., Yue, M., and Shahabi, C. Hagen: Homophily-aware graph convolutional recurrent network for crime forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4193–4200, 2022.

Wang, G., Du, M., Liu, N., Zou, N., and Hu, X. Mitigating algorithmic bias with limited annotations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 241–258. Springer, 2023a.

Wang, Z., Jia, Z., Zheng, S., Zhang, Z., Fu, X., Ng, T. S. E., and Wang, Y. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023b.

Wang, Z., Xu, Z., Wu, X., and Ng, T. S. E. Cupcake: A compression optimizer for scalable communication-efficient distributed training. In *Proceedings of Machine Learning and Systems*, 2023c.

Xie, J., Liu, Y., and Shen, Y. Explaining dynamic graph neural networks via relevance back-propagation. *arXiv preprint arXiv:2207.11175*, 2022.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Yao, Y., Wang, P., Tian, B., Cheng, S., Li, Z., Deng, S., Chen, H., and Zhang, N. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*, 2023.

Yu, C., Jeoung, S., Kasi, A., Yu, P., and Ji, H. Unlearning bias in language models by partitioning gradients. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 6032–6048, Toronto, Canada, July 2023. Association for Computational Linguistics.

Yu, J., Wang, J., Zhao, H., Gao, J., Kang, Y., Cao, D., Wang, Z., and Hou, T. Organic compound synthetic accessibility prediction based on the graph attention mechanism. *Journal of Chemical Information and Modeling*, 62(12): 2973–2986, 2022.

Zhang, N., Yao, Y., Tian, B., Wang, P., Deng, S., Wang, M., Xi, Z., Mao, S., Zhang, J., Ni, Y., Cheng, S., Xu, Z., Xu, X., Gu, J.-C., Jiang, Y., Xie, P., Huang, F., Liang, L., Zhang, Z., Zhu, X., Zhou, J., and Chen, H. A comprehensive study of knowledge editing for large language models, 2024a.

Zhang, N., Yao, Y., Tian, B., Wang, P., Deng, S., Wang, M., Xi, Z., Mao, S., Zhang, J., Ni, Y., et al. A comprehensive study of knowledge editing for large language models. *arXiv preprint arXiv:2401.01286*, 2024b.

Zhao, J., Mostafa, H., Galkin, M., Bronstein, M., Zhu, Z., and Tang, J. Graphany: A foundation model for node classification on any graph, 2024.

Zhou, K., Huang, X., Li, Y., Zha, D., Chen, R., and Hu, X. Towards deeper graph neural networks with differentiable group normalization. *Advances in neural information processing systems*, 33:4917–4928, 2020.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021.

Zhou, K., Choi, S.-H., Liu, Z., Liu, N., Yang, F., Chen, R., Li, L., and Hu, X. Adaptive label smoothing to regularize large-scale graph training. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pp. 55–63. SIAM, 2023.

Zhu, C., Rawat, A. S., Zaheer, M., Bhojanapalli, S., Li, D., Yu, F., and Kumar, S. Modifying memories in transformer models, 2020.

A. Extended Preliminary and Proposed Method

A.1. The Data Management

Here, we provide some practical guidance in terms of the data management of graph editing tasks. Suppose a model M_0 is expecting to do n sequential edits within a graph dataset D. In an inductive setting, we should have three disjoint graphs D_{train}, D_{val}, and D_{test} (Hamilton et al., 2017). **In practice, it is recommended to have the edited targets $\{e_1, ..., e_n\}$ prefetched from the D_{val}** (where $M_0(e_i) \neq Y_{e_i}$, as all editing targets need to be wrongly predicted by M to be eligible), so that one can easily evaluate the overall test accuracy of a model $M(D_{\text{test}})$ before and after edits without further modification[2].

One important but often overlooked issue under the sequential edit paradigm is how to handle "accidentally corrected" edit targets. Because $\{e_1..., e_n\}$ is prefetched from D_{val} where $M_0(e_i) \neq Y_{e_i}$, it is possible that when conducting $\texttt{Edit}(M_0, e_1) = M_1$, e_2 is "accidentally" made correct — i.e., $M_1(e_{50}) = Y_{e_2}$. In this case, the model M_1 should not perform any further option but skip e_2, as e_2 is not considered an error to M_1 and therefore can't trigger an `Edit` operation. Many previous work, including EGNN (Liu et al., 2023a), simply loop over $\{e_1..., e_n\}$ and perform `Edit` operations sequentially until all n edits are conducted. This setup is not a true reflection of how model maintenance is conducted in the real world, as a model should not conduct `Edit` operation without a specific high-profile error in the first place.

A.2. Formal Procedure of Edit-aware Training Sample Mix-up

Algorithm 1 SEED-GNN Training Sample Mix-up Selection (§5.2)

Input: an edit target e_i, a set of previously edited samples $E = e_1, e_2, \ldots$, a set of training samples D_{train}, α, β.
Initialize: Editing batch of samples, $\mathcal{E} = \{e_i\}$, to update the model.
$\mathcal{E} = \mathcal{E} \cup E$ {Append all previously edited targets to the current editing batch.}
$\mathcal{N}(e_i) = \{e_j | e_j \in D_{\text{train}} \wedge \text{dist}(e_i, e_j) \leq k\}$ {The set of all k-hops neighbors of edit target.}
for $n_i \in \alpha * \beta$ randomly selected samples from $\mathcal{N}(e_i)$ **do**
  $\mathcal{E} = \mathcal{E} \cup \{n_i\}$ {Append $\alpha * \beta$ k-hop neighbors of e_i to the editing batch.}
end for
for $r_i \in (1 - \alpha) * \beta$ randomly selected samples from D_{train} **do**
  $\mathcal{E} = \mathcal{E} \cup \{r_i\}$ {Append $(1 - \alpha) * \beta$ randomly selected training samples to the editing batch.}
end for
return $\mathcal{E}$

A.3. More Limitations

Given the close connections from fair learning, locality-generality trade-off, and maybe even explainable GNNs to model editing — which are some mature graph learning subfields — we recommend that future scholars explore such areas for insights (Lina & Silva, 2024; Wang et al., 2023a; Chang et al., 2023; Liu, 2023; Xie et al., 2022; Rong et al., 2023). Moreover, although model editing is often treated as a post-hoc operation applicable to any trained model, whether the editing effect would change depending on the training recipe or setup of the pre-edit model remains unexplored. This is especially important given the popularity of different GNN training techniques, as well as the system support works (Wang et al., 2023c;b) that come with them to make them trained faster (Liu et al., 2021; 2023b), larger (Chen et al., 2022; Zhou et al., 2020; 2021; 2023), and more versatile (Sun et al., 2022; Zhao et al., 2024). In this work, we only evaluated our method against four established GNN architectures trained in a vanilla fashion, leaving its compatibility with more modern GNNs unanswered.

[2]The alternative setup is to select editing targets from D_{test}, but then the test accuracy should be evaluated on $D_{\text{test}} \backslash \{e_1, e_2, \ldots, e_n\}$, complicating the pipeline. This setup is not recommended unless the D_{val} is already otherwise utilized.

B. Ablation Studies

In this section, we investigate the influence of various SEED-GNN-specific hyperparameters and designs: α, β (§5.2) and incremental batching (§5.3). Here, we first report the hyperparameter settings of SEED-GNN in Table 7. We note the following non-SEED-GNN-specific hyperparameter settings are all copied from EGNN by Liu et al. (2023a) for better alignment except LR (where EGNN uses LR=0.01 for most experiments) — as we found this smaller LR to be better for our proposed method (Table 11).

Table 7. Training hyperparameters / configurations of our proposed method. "Steps" is the maximum allowed edit steps per edit target.

Backbone	Dataset	#Layers	#Hidden Channels	LR	Dropout	Epochs	α	β	Steps
GCN	Cora	2	32	0.001	0.1	200	0.25	100	500
	A-computers	2	32	0.001	0.1	400			
	A-photo	2	32	0.001	0.1	400			
	Coauthor-CS	2	32	0.001	0.1	400			
	ogbn-arxiv	3	128	0.001	0.5	500			
	ogbn-products	3	256	0.001	0.5	500			
Graph-SAGE	Cora	2	32	0.001	0.1	200	0.25	100	500
	A-computers	2	32	0.001	0.1	400			
	A-photo	2	32	0.001	0.1	400			
	Coauthor-CS	2	32	0.001	0.1	400			
	ogbn-arxiv	3	128	0.001	0.5	500			
	ogbn-products	3	256	0.001	0.5	500			
GIN	Cora	2	32	0.001	0.1	200	0.25	100	500
	A-computers	2	32	0.001	0.1	400			
	A-photo	2	32	0.001	0.1	400			
	Coauthor-CS	2	32	0.001	0.1	400			
	ogbn-arxiv	3	128	0.001	0.5	500			
	ogbn-products	3	256	0.001	0.5	500			
GAT	Cora	2	32	0.01	0.1	200	0.25	100	500
	A-computers	2	32	0.01	0.1	400			
	A-photo	2	32	0.01	0.1	400			
	Coauthor-CS	2	32	0.01	0.1	400			
	ogbn-arxiv	3	128	0.01	0.5	500			
	ogbn-products	3	256	0.01	0.5	500			

B.1. Influence of SEED-GNN-specific Hyperparameters and Designs

Table 8. Ablation Study of Hyperparameter α (a.k.a. ratio of edit-aware samples in editing batch, $\beta = 100$)

Backbone (PE Acc.)	Method	Test Drawdown (Success Rate) 1th	10th	25th	50th	Max DD	Avg DD	Avg SR
Cora 2,485 Nodes 5,069 Edges 7 Classes 1,433 Features								
GCN (89.80%)	$\alpha = 0$	0.2 (1.0)	1.8 (1.0)	2.4 (1.0)	6.4 (1.0)	6.8	4.00	0.98
	$\alpha = 0.25$	0.2 (1.0)	3.2 (1.0)	3.2 (1.0)	8.0 (1.0)	8.0	4.1	1.0
	$\alpha = 0.50$	0.6 (1.0)	3.0 (1.0)	3.0 (0.96)	7.8 (1.0)	7.8	4.9	1.0
	$\alpha = 0.75$	1.4 (1.0)	3.4 (1.0)	3.4 (1.0)	6.4 (1.0)	8.4	4.9	1.0
	$\alpha = 1.0$	0.8 (1.0)	1.6 (1.0)	4.6 (1.0)	6.8 (1.0)	13.4	5.9	1.0
Graph-SAGE (86.60%)	$\alpha = 0$	-0.4 (0.0)	0.6 (1.0)	1.8 (1.0)	2.6 (0.98)	4.0	2.10	0.96
	$\alpha = 0.25$	0.8 (1.0)	-0.8 (1.0)	2.6 (1.0)	4.2 (1.0)	5.0	2.1	1.0
	$\alpha = 0.50$	0.4 (1.0)	0.2 (1.0)	2.0 (1.0)	4.0 (1.0)	4.0	1.7	1.0
	$\alpha = 0.75$	0.0 (1.0)	0.8 (1.0)	3.2 (1.0)	3.8 (1.0)	5.8	2.7	1.0
	$\alpha = 1.0$	-0.2 (1.0)	1.0 (1.0)	1.8 (1.0)	5.0 (1.0)	6.6	3.2	1.0
ogbn-arxiv 169,343 Nodes 1,166,243 Edges 40 Classes 128 Features								
GCN (93.81%)	$\alpha = 0$	1.7 (0.0)	1.5 (0.2)	0.9 (0.24)	1.2 (0.12)	6.4	2.20	0.26
	$\alpha = 0.25$	1.6 (1.0)	3.2 (1.0)	8.0 (1.0)	7.6 (1.0)	27.8	10.0	0.999
	$\alpha = 0.50$	4.0 (1.0)	12.3 (1.0)	8.9 (0.96)	18.7 (0.98)	24.6	12.5	0.992
	$\alpha = 0.75$	3.2 (1.0)	10.0 (1.0)	8.7 (1.0)	10.9 (1.0)	19.6	8.7	0.994
	$\alpha = 1.0$	5.7 (1.0)	7.2 (1.0)	10.0 (1.0)	26.8 (1.0)	26.8	11.3	0.997
Graph-SAGE (86.60%)	$\alpha = 0$	1.5 (1.0)	0.7 (0.2)	1.2 (0.12)	1.5 (0.26)	9.3	2.30	0.27
	$\alpha = 0.25$	1.1 (1.0)	6.0 (1.0)	10.4 (1.0)	7.4 (0.96)	19.3	8.6	0.996
	$\alpha = 0.50$	2.2 (1.0)	20.0 (1.0)	7.8 (1.0)	10.4 (1.0)	20.0	9.4	0.997
	$\alpha = 0.75$	1.7 (1.0)	6.0 (1.0)	6.2 (1.0)	10.5 (1.0)	22.5	10.6	0.994
	$\alpha = 1.0$	0.8 (1.0)	6.2 (1.0)	7.7 (1.0)	11.6 (1.0)	19.7	9.9	0.999

Table 9. Ablation Study of with different Hyperparameter β (a.k.a. number of extra sample to be included in editing batch, $\alpha = 0.25$)

Backbone (PE Acc.)	Method	Test Drawdown (Success Rate) 1th	10th	25th	50th	Max DD	Avg DD	Avg SR
Cora 2,485 Nodes 5,069 Edges 7 Classes 1,433 Features								
GCN (89.80%)	$\beta = 100$	0.2 (1.0)	3.2 (1.0)	3.2 (1.0)	8.0 (1.0)	8.0	4.1	1.0
	$\beta = 150$	0.6 (1.0)	3.0 (1.0)	2.4 (1.0)	5.6 (1.0)	7.4	4.2	1.0
	$\beta = 300$	0.4 (1.0)	2.8 (1.0)	3.0 (1.0)	5.4 (1.0)	7.2	4.1	1.0
	$\beta = 500$	0.2 (1.0)	3.4 (1.0)	5.8 (1.0)	6.8 (1.0)	8.6	5.50	1.00
	$\beta = 700$	-0.2 (1.0)	3.4 (1.0)	4.4 (1.0)	7.2 (1.0)	7.4	4.80	1.00
	$\beta = 800$	0.0 (1.0)	3.4 (1.0)	4.2 (1.0)	7.8 (1.0)	7.8	4.90	1.00
Graph-SAGE (86.60%)	$\beta = 100$	0.8 (1.0)	-0.8 (1.0)	2.6 (1.0)	4.2 (1.0)	5.0	2.1	1.0
	$\beta = 150$	0.2 (1.0)	1.2 (1.0)	2.2 (0.96)	4.8 (1.0)	4.8	2.6	1.0
	$\beta = 300$	1.4 (1.0)	3.4 (1.0)	3.4 (1.0)	6.4 (1.0)	8.4	4.9	1.0
	$\beta = 500$	-1.4 (1.0)	2.8 (1.0)	2.4 (1.0)	3.6 (1.0)	5.2	2.50	1.00
	$\beta = 700$	-0.8 (1.0)	3.0 (1.0)	2.4 (1.0)	4.0 (1.0)	5.0	2.60	1.00
	$\beta = 800$	-0.6 (1.0)	2.8 (1.0)	2.6 (1.0)	4.2 (1.0)	5.2	2.80	1.00
ogbn-arxiv 169,343 Nodes 1,166,243 Edges 40 Classes 128 Features								
GCN (89.80%)	$\beta = 100$	1.6 (1.0)	3.2 (1.0)	8.0 (1.0)	7.6 (1.0)	27.8	10.0	0.999
	$\beta = 150$	2.9 (1.0)	3.0 (1.0)	3.5 (0.96)	6.4 (1.0)	13.4	6.6	0.996
	$\beta = 300$	-0.2 (1.0)	2.4 (1.0)	3.4 (1.0)	4.4 (1.0)	5.0	3.2	1.0
	$\beta = 500$	9.6 (0.0)	9.4 (1.0)	11.7 (0.92)	0.8 (0.1)	12.6	6.10	0.67
	$\beta = 700$	8.9 (1.0)	13.7 (1.0)	0.9 (0.08)	0.6 (0.06)	13.7	2.60	0.32
	$\beta = 800$	6.7 (1.0)	9.1 (1.0)	3.8 (0.52)	4.3 (0.46)	19.3	5.90	0.69
Graph-SAGE (86.60%)	$\beta = 100$	1.1 (1.0)	6.0 (1.0)	10.4 (1.0)	7.4 (0.96)	19.3	8.6	0.996
	$\beta = 150$	2.0 (1.0)	10.2 (1.0)	6.2 (1.0)	10.0 (1.0)	15.5	7.4	0.998
	$\beta = 300$	1.4 (1.0)	14.0 (1.0)	1.8 (1.0)	-0.6 (1.0)	18.4	3.5	0.32
	$\beta = 500$	0.8 (1.0)	5.7 (1.0)	16.6 (1.0)	2.2 (0.42)	17.3	7.90	0.91
	$\beta = 700$	0.6 (1.0)	0.4 (0.1)	0.8 (0.08)	0.5 (0.04)	17.6	2.50	0.25
	$\beta = 800$	1.4 (1.0)	7.3 (1.0)	1.0 (0.08)	0.8 (0.08)	16.3	3.30	0.41

Table 10. Ablation Study of Incremental Batching (§5.3)

Backbone (PE Acc.)	Method	Test Drawdown (Success Rate) 1th	10th	25th	50th	Max DD	Avg DD	Avg SR
Cora 2,485 Nodes 5,069 Edges 7 Classes 1,433 Features								
GCN (89.8%)	with $\alpha = 0.25$, $\beta = 100$ (§5.2)	0.2 (1.0)	0.2 (0.2)	0.2 (0.12)	0.2 (0.1)	0.2	0.2	0.16
	with incremental batching (§5.3)	0.2 (1.0)	3.2 (1.0)	3.2 (1.0)	8.0 (1.0)	8.0	4.1	1.0
Graph-SAGE (86.6%)	with $\alpha = 0.25$, $\beta = 100$	0.8 (1.0)	0.8 (0.3)	0.8 (0.12)	0.8 (0.16)	0.8	0.8	0.20
	with incremental batching	0.8 (1.0)	-0.8 (1.0)	2.6 (1.0)	4.2 (1.0)	5.0	2.1	1.0
ogbn-arxiv 169,343 Nodes 1,166,243 Edges 40 Classes 128 Features								
GCN (70.25%)	with $\alpha = 0.25$, $\beta = 100$	1.6 (1.0)	3.3 (0.1)	(2.2, 0.12)	4.0 (0.12)	11.6	3.60	0.183
	with incremental batching	1.6 (1.0)	3.2 (1.0)	8.0 (1.0)	7.6 (1.0)	27.8	10.0	0.99
Graph-SAGE (68.45%)	with $\alpha = 0.25$, $\beta = 100$	1.1 (1.0)	1.3 (0.2)	1.3 (0.12)	1.3 (0.08)	6.8	1.4	0.18
	with incremental batching	1.1 (1.0)	6.0 (1.0)	10.4 (1.0)	7.4 (1.0)	19.3	8.6	0.996

From Table 8 and Table 9, we can observe that adding a small number of training samples (e.g., $\beta = 100$) as well as a proper mixture of neighborhood targets (e.g., $\alpha = 0.25$) to the editing batch significantly improve the performance of reported metrics (where non-batched baseline results can be found in Table 3). Additionally, Table 10 shows the adaptation of incremental batching directly boosts the overall Success Rate of editing.

B.2. Influence of General Hyperparameters

Table 11. Ablation Study of Learning Rates

Backbone	LR	Test Drawdown (Success Rate) 1th	10th	25th	50th	Max DD	Avg DD	Avg SR
	Cora	2,485 Nodes	5,069 Edges	7 Classes	1,433 Features			
GCN	0.001	0.6 (1.0)	2.4 (1.0)	3.2 (1.0)	2.6 (1.0)	5.4	3.2	1.000
	0.005	0.0 (1.0)	0.6 (1.0)	3.6 (1.0)	4.8 (1.0)	6.4	3.6	1.000
	0.01	0.2 (1.0)	3.2 (1.0)	3.2 (1.0)	8.0 (1.0)	8.0	4.1	1.000
	0.05	1.4 (1.0)	6.0 (1.0)	5.6 (1.0)	13.4 (1.0)	13.8	7.5	1.000
	0.1	2.0 (1.0)	6.0 (1.0)	8.8 (1.0)	6.2 (1.0)	15.4	8,9	1.000
Graph-SAGE	0.001	-0.4 (1.0)	-0.8 (1.0)	0.2 (1.0)	1.0 (1.0)	3.0	1.0	1.000
	0.005	1.2 (1.0)	-1.0 (1.0)	1.8 (1.0)	2.4 (1.0)	4.4	1.2	1.000
	0.01	0.8 (1.0)	-0.8 (1.0	2.6 (1.0)	4.2 (1.0)	5.0	2.1	1.000
	0.05	0.6 (1.0)	0.8 (1.0)	5.0 (1.0)	4.4 (1.0)	8.0	3.0	1.000
	0.1	-0.2 (1.0)	1.0 (1.0)	5.6 (1.0)	17.0 (1.0)	25.6	10.5	0.996
GIN	0.001	-1.0 (0.0)	0.0 (0.9)	1.0 (0.96)	0.4 (1.0)	1.4	0.2	0.909
	0.005	-0.6 (1.0)	0.4 (1.0)	0.6 (1.0)	0.4 (1.0)	3.6	0.6	0.977
	0.01	-0.6 (1.0)	-0.2 (1.0)	1.6 (1.0)	1.0 (1.0)	4.6	0.7	1.000
	0.05	-0.8 (1.0)	1.6 (1.0)	1.4 (0.96)	2.9 (1.0)	4.2	0.9	0.994
	0.1	2.2 (1.0)	7.4 (0.8)	2.6 (0.36)	2.6 (0.26)	13.0	3.2	0.490
	ogbn-arxiv	169,343 Nodes	1,166,243 Edges	40 Classes	128 Features			
GCN	0.001	3.7 (1.0)	4.7 (1.0)	5.3 (1.0)	6.2 (1.0)	9.8	6.1	0.999
	0.005	3.2 (1.0)	2.5 (1.0)	7.3 (1.0)	8.4 (1.0)	12.8	6.3	0.999
	0.01	1.6 (1.0)	3.2 (1.0)	8.0 (1.0)	7.6 (1.0)	27.8	10.0	0.999
	0.05	7.4 (0.0)	3.4 (0.6)	1.1 (0.36)	1.1 (0.18)	9.7	2.7	0.251
	0.1	9.1 (0.0)	2.3 (0.1)	2.1 (0.28)	1.5 (0.22)	9.1	2.4	0.210
Graph-SAGE	0.001	0.1 (1.0)	5.8 (1.0)	5.5 (1.0)	4.8 (1.0)	11.0	4.9	1.000
	0.005	1.1 (1.0)	10.1 (1.0)	6.8 (1.0)	5.8 (1.0)	13.2	6.4	0.999
	0.01	1.1 (1.0)	6.0 (1.0)	10.4 (1.0)	7.4 (1.0)	19.3	8.6	0.996
	0.05	1.1 (1.0)	0.7 (0.3)	1.8 (0.12)	0.3 (0.1)	5.0	1.5	0.172
	0.1	2.0 (1.0)	1.3 (0.2)	3.5 (0.12)	1.3 (0.14)	5.4	1.8	0.186
GIN	0.001	0.6 (1.0)	1.4 (0.9)	7.3 (0.96)	8.9 (1.0)	9.4	5.5	0.949
	0.005	0.7 (0.0)	10.4 (0.9)	7.5 (0.96)	4.9 (0.22)	10.7	5.6	0.742
	0.01	0.4 (1.0)	5.3 (1.0)	7.8 (1.0)	9.8 (1.0)	14.6	7.9	0.998
	0.05	0.2 (1.0)	3.0 (0.4)	4.6 (0.24)	5.4 (0.24)	11.7	4.6	0.255
	0.1	1.5 (1.0)	2.9 (0.2)	6.4 (0.28)	9.0 (0.28)	9.0	4.5	0.255

C. System Reports

In this section, we report the runtime and memory usage of different editing methods. It is observable that EGNN (Liu et al., 2023a) and SEED-GNN are often among the most efficient methods due to the frozen GNN + active MLP ensemble design. Where SEED-GNN shows a slight advantage on a few setups in terms of total times due to being able to patch editing targets with fewer steps (Table 7). We also note the runtime of EGNN and SEED-GNN are based on cached GNN output utilizing the property illustrated in Section 5.1, where the end-to-end runtime shall add the full `forward()` of all input cases in the GNN part of the methods.

Table 12. Time / Memory of Editing Methods on ogbn-products

Backbone	Method	Editing Time (s) 1th	10th	25th	50th	Total Time (s)	Peak GPU Memory (MB)
	ogbn-products	2,449,029 Nodes	61,859,140 Edges	47 Classes	218 Features		
GCN	FT	2.36	3.74	5.58	0.00	202.64	26429.04
	ENN	2344.07	2343.33	2343.62	2343.63	110156.88	39708.80
	EGNN	0.06	0.04	0.03	0.07	2.57	5753.07
	Adapter	12.0	29.82	38.79	20.89	586.23	25410.13
	LoRA	42.916	15.084	42.834	16.911	982.36	27726.438
	SEED-GNN	0.033	0.047	0.00	0.065	1.442	5755.685
Graph-SAGE	FT	2.94	2.80	0.00	4.17	141.15	27358.56
	ENN	1897.08	1897.12	1896.88	1896.58	91220.18	41208.87
	EGNN	0.06	0.04	0.06	0.03	1.80	4163.43
	Adapter	5.53	21.82	20.47	32.70	519.67	26722.59
	LoRA	24.15	1.49	0.00	2.89	595.80	28707.05
	SEED-GNN	0.043	0.047	0.082	0.000	1.978	4176.591
GIN	FT	4.22	5.88	7.49	7.49	273.25	64030.91
	EGNN	0.04	0.05	0.07	0.05	2.40	4167.94
	Adapter	11.75	14.49	15.92	8.76	715.10	49195.40
	SEED-GNN	0.034	0.054	0.080	0.120	2.599	4023.822

Table 13. Time / Memory of Editing Methods on Cora

Backbone	Method	Editing Time (s) 1th	10th	25th	50th	Total Time (s)	Peak GPU Memory (MB)
	Cora	2,485 Nodes	5,069 Edges	7 Classes	1,433 Features		
GCN	FT	0.01	0.01	0.00	0.01	0.44	43.6
	ENN	4.46	4.45	4.43	4.45	223.67	44.84
	EGNN	0.01	0.00	0.06	0.00	1.75	43.32
	Adapter	0.16	0.00	0.02	0.01	1.28	44.20
	LoRA	0.01	0.01	0.01	0.01	1.47	43.71
	SEED-GNN	0.006	0.008	0.020	0.000	0.504	45.952
Graph-SAGE	FT	0.02	0.00	0.02	0.01	0.58	59.63
	ENN	6.27	6.04	6.15	6.27	305.18	75.99
	EGNN	0.03	0.01	0.02	0.07	1.75	43.42
	Adapter	0.04	0.09	0.02	0.00	1.59	58.102
	LoRA	0.02	0.06	0.00	0.02	1.80	59.61
	SEED-GNN	0.037	0.005	0.010	0.033	0.931	53.232
GAT	FT	0.02	0.00	0.02	0.03	1.13	103.45
	ENN	12.92	12.76	12.89	12.88	641.52	156.76
	EGNN	0.02	0.00	0.03	0.17	2.02	45.53
	Adapter	0.06	0.118	0.00	0.046	3.02	77.37
	SEED-GNN	0.020	0.000	0.000	0.007	0.806	79.286
GIN	FT	0.02	0.00	0.04	0.01	0.97	86.86
	ENN	15.60	15.11	15.21	15.12	759.47	112.69
	EGNN	0.06	0.00	0.03	0.03	3.04	43.50
	Adapter	0.05	0.02	0.02	0.02	1.06	86.50
	SEED-GNN	0.063	0.011	0.000	0.010	1.298	48.182

D. Extended Experiments and Results

D.1. Additional SEED-GNN Variants

We compare a few reviewer-suggested approaches, where most of them are different architectures taken on full or partial SEED-GNN ingredients (as our batching design is architecture-agnostic). We observe that FT-SEED-GNN is the only competitive method. However, it sometimes fell short of the Success Rate department to SEED-GNN — which is an important aspect of an editing method as explained in Section 4.2. Moreover, since FT-SEED-GNN is based upon the full model finetune pipeline, it is naturally a lot more resource-intensive than SEED-GNN, evidenced by readings in Appendix C; which is also a major disadvantage for being an editing method as discussed in Section 5.1.

Table 14. Ablation Study of different editing methods with SEED-GNN ingredients on large graphs. "FT-SEED-GNN" is vanilla fine-tuning the model with SEED-GNN ingredients (edit-aware training samples mixup, incremental batching, and stoppage). "ENN" is ENN (Sinitsin et al., 2020) with SEED-GNN ingredients. "batched-EGNN" is EGNN (Liu et al., 2023a) training sample mixup. "EGNN-50" is EGNN but editing all 50 edits at once.

Backbone (PE Acc.)	**Method**	**Test Drawdown (Success Rate)** 1th	10th	25th	50th	**Max DD**	**Avg DD**	**Avg SR**
	Amazon Computers	13,381 Nodes	245,778 Edges	10 Classes	767 Features			
GCN (85.77%)	FT-SEED-GNN	0.5 (0.0)	2.3 (0.7)	0.7 (0.96)	1.3 (0.96)	3.1	1.50	0.89
	ENN-SEED-GNN	31.9 (1.0)	69.0 (0.1)	69.0 (0.08)	69.0 (0.12)	69.0	68.30	0.13
	batched-EGNN	-0.9 (1.0)	-0.6 (0.4)	-0.6 (0.32)	-0.6 (0.26)	-0.6	-0.60	0.37
	EGNN-50	-	-	-	58.1 (0.98)	-	-	-
	SEED-GNN	0.6 (1.0)	-0.9 (1.0)	-0.1 (1.0)	2.7 (1.0)	3.3	1.1	0.99
Graph-SAGE (83.23%)	FT-SEED-GNN	0.5 (0.0)	2.3 (0.7)	0.7 (0.96)	1.3 (0.96)	3.1	1.50	0.89
	ENN-SEED-GNN	67.4 (1.0)	74.1 (0.0)	83.1 (0.0)	83.1 (0.06)	83.1	80.00	0.04
	batched-EGNN	-0.9 (0.0)	-0.9 (0.1)	-0.9 (0.12)	-0.9 (0.12)	-0.9	-0.90	0.11
	EGNN-50	-	-	-	38.1 (1.0)	-	-	-
	SEED-GNN	1.9 (1.0)	0.5 (1.0)	-0.9 (1.0)	-0.2 (1.0)	3.5	0.0	1.0
GIN (66.11%)	FT-SEED-GNN	-2.8 (1.0)	-10.8 (1.0)	-15.0 (0.96)	-19.7 (0.98)	-2.8	-11.60	0.93
	ENN-SEED-GNN	35.5 (0.0)	40.1 (0.3)	63.2 (0.16)	31.6 (0.12)	66.0	43.80	0.14
	batched-EGNN	-3.5 (1.0)	-3.5 (0.3)	-3.5 (0.2)	-3.5 (0.18)	-3.5	-3.50	0.28
	EGNN-50	-	-	-	10.8 (0.98)	-	-	-
	SEED-GNN	-3.1 (0.0)	-4.1 (1.0)	-9.6 (1.0)	-10.2 (1.0)	3.1	-8.0	0.95
	ogbn-arxiv	169,343 Nodes	1,166,243 Edges	40 Classes	128 Features			
GCN (70.26%)	FT-SEED-GNN	3.7 (0.0)	3.3 (0.7)	2.7 (1.0)	3.6 (1.0)	6.6	3.60	0.89
	ENN-SEED-GNN	12.2 (0.0)	67.3 (0.0)	67.3 (0.0)	66.8 (0.04)	69.0	65.30	0.01
	batched-EGNN	4.0 (1.0)	4.0 (0.2)	11.1 (0.16)	4.2 (0.18)	11.1	4.70	0.23
	EGNN-50	-	-	-	42.2 (0.96)	-	-	-
	SEED-GNN	-3.7 (1.0)	4.7 (1.0)	5.3 (1.0)	6.2 (1.0)	9.8	6.1	1.0
Graph-SAGE (68.45%)	FT-SEED-GNN	3.7 (0.0)	3.3 (0.7)	2.7 (1.0)	3.6 (1.0)	6.6	3.60	0.89
	ENN-SEED-GNN	18.9 (0.0)	46.9 (0.0)	46.9 (0.0)	62.6 (0.12)	66.4	53.60	0.06
	batched-EGNN	1.4 (1.0)	1.4 (0.1)	1.4 (0.04)	1.4 (0.04)	1.4	1.40	0.10
	EGNN-50	-	-	-	32.7 (1.0)	-	-	-
	SEED-GNN	0.1 (1.0)	5.8 (1.0)	5.5 (1.0)	4.8 (1.0)	11.0	4.9	1.0
GIN (66.17%)	FT-SEED-GNN	0.6 (0.0)	4.9 (1.0)	7.0 (1.0)	5.0 (0.9)	12.6	6.40	0.90
	ENN-SEED-GNN	60.3 (0.0)	62.2 (0.2)	51.2 (0.16)	51.8 (0.14)	65.6	54.50	0.14
	batched-EGNN	2.5 (1.0)	3.9 (0.3)	4.4 (0.24)	4.4 (0.18)	4.4	4.00	0.26
	EGNN-50	-	-	-	51.4 (0.86)	-	-	-
	SEED-GNN	0.7 (1.0)	1.1 (0.8)	4.9 (0.96)	6.7 (0.98)	8.6	5.2	0.945
	ogbn-products	2,449,029 Nodes	61,859,140 Edges	47 Classes	218 Features			
GCN (74.90%)	FT-SEED-GNN	0.4 (1.0)	5.6 (1.0)	10.1 (1.0)	5.7 (1.0)	10.1	5.20	0.99
	ENN-SEED-GNN	1.7 (0.0)	52.4 (0.1)	47.9 (0.16)	47.9 (0.14)	52.4	44.80	0.11
	batched-EGNN	2.2 (1.0)	2.2 (0.3)	7.6 (0.28)	7.6 (0.26)	9.8	6.20	0.27
	EGNN-50	-	-	-	55.3 (1.0)	-	-	-
	SEED-GNN	3.4 (1.0)	7.7 (1.0)	7.0 (1.0)	10.7 (1.0)	12.4	8.3	1.0
Graph-SAGE (76.37%)	FT-SEED-GNN	0.9 (1.0)	5.3 (1.0)	4.5 (0.96)	6.8 (1.0)	8.8	4.90	0.99
	ENN-SEED-GNN	0.7 (1.0)	68.2 (0.2)	70.8 (0.12)	67.5 (0.14)	71.7	61.90	0.19
	batched-EGNN	2.0 (1.0)	2.0 (0.3)	2.0 (0.2)	5.0 (0.26)	5.6	2.90	0.25
	EGNN-50	-	-	-	41.0 (1.0)	-	-	-
	SEED-GNN	1.8 (1.0)	2.5 (1.0)	32. (1.0)	6.2 (1.0)	6.6	3.6	1.0
GIN (65.79%)	FT-SEED-GNN	4.7 (0.0)	15.3 (1.0)	17.9 (0.96)	14.6 (0.98)	20.8	15.50	0.92
	ENN-SEED-GNN				OOM			
	batched-EGNN	0.2 (1.0)	0.2 (0.1)	0.2 (0.08)	0.2 (0.06)	0.2	0.20	0.12
	EGNN-50	-	-	-	54.5 (0.98)	-	-	-
	SEED-GNN	-0.3 (0.0)	1.4 (0.9)	3.5 (0.96)	1.8 (1.0)	4.3	2.3	0.93

Table 15. Ablation Study of different editing methods with SEED-GNN ingredients on small graphs. "FT-SEED-GNN" is vanilla fine-tuning the model with SEED-GNN ingredients (edit-aware training samples mixup, incremental batching, and stoppage). "ENN" is ENN (Sinitsin et al., 2020) with SEED-GNN ingredients. "batched-EGNN" is EGNN (Liu et al., 2023a) training sample mixup. "EGNN-50" is EGNN but editing all 50 edits at once.

Backbone (PE Acc.)	**Method**	**Test Drawdown (Success Rate)** 1th	10th	25th	50th	**Max DD**	**Avg DD**	**Avg SR**
	Cora	2,485 Nodes	5,069 Edges	7 Classes	1,433 Features			
GCN (89.80%)	FT-SEED-GNN	-0.2 (0.0)	0.4 (0.9)	1.2 (0.96)	0.6 (0.98)	1.6	0.60	0.90
	ENN-SEED-GNN	0.6 (1.0)	5.4 (0.9)	7.4 (0.96)	13.6 (0.98)	13.8	8.00	0.95
	batched-EGNN	0.2 (1.0)	0.2 (0.1)	0.2 (0.08)	0.2 (0.06)	0.2	0.20	0.11
	EGNN-50	-	-	-	9.6 (1.0)	-	-	-
	SEED-GNN	0.6 (1.0)	2.4 (1.0)	3.2 (1.0)	2.6 (1.0)	5.4	3.2	1.0
Graph-SAGE (86.60%)	FT-SEED-GNN	-0.2 (0.0)	0.4 (0.9)	1.2 (0.96)	0.6 (0.98)	1.6	0.60	0.90
	ENN-SEED-GNN	0.0 (1.0)	48.0 (1.0)	42.2 (1.0)	31.2 (0.98)	51.4	40.90	0.90
	batched-EGNN	-0.2 (1.0)	-0.2 (0.2)	-0.2 (0.08)	-0.2 (0.1)	-0.2	-0.20	0.14
	EGNN-50	-	-	-	8.6 (1.0)	-	-	-
	SEED-GNN	-0.4 (1.0	-0.8 (1.0)	0.2 (1.0)	1.0 (1.0)	3.0	1.0	1.0
GIN (84.2%)	FT-SEED-GNN	-0.8 (1.0)	-3.4 (1.0)	-2.4 (1.0)	-1.4 (0.98)	0.4	-1.80	0.97
	ENN-SEED-GNN	9.2 (1.0)	39.4 (0.4)	47.4 (0.8)	48.6 (0.76)	63.0	48.80	0.55
	batched-EGNN	0.8 (1.0)	0.8 (0.1)	-2.4 (0.2)	-2.4 (0.12)	0.8	-1.20	0.17
	EGNN-50	-	-	-	9.6 (0.96)	-	-	-
	SEED-GNN	-0.4 (0.0)	0.2 (0.9)	1.2 (1.0)	1.6 (1.0)	2.6	0.70	0.90
	Amazon Photo	7,487 Nodes	119,043 Edges	8 Classes	745 Features			
GCN (93.81%)	FT-SEED-GNN	1.5 (1.0)	4.8 (0.9)	3.7 (0.96)	8.6 (0.98)	8.6	3.20	0.93
	ENN-SEED-GNN	0.8 (1.0)	63.9 (0.6)	64.0 (0.28)	64.0 (0.18)	64.0	62.50	0.34
	batched-EGNN	-0.4 (1.0)	-0.4 (0.4)	-0.4 (0.24)	-0.4 (0.24)	-0.4	-0.40	0.31
	EGNN-50	-	-	-	57.9 (1.0)	-	-	-
	SEED-GNN	-0.1 (1.0)	1.2 (1.0)	1.1 (1.0)	1.1 (1.0)	1.8	0.8	1.0
Graph-SAGE (94.36%)	FT-SEED-GNN	1.5 (1.0)	4.8 (0.9)	3.7 (0.96)	8.6 (0.98)	8.6	3.20	0.93
	ENN-SEED-GNN	24.6 (0.0)	93.9 (0.0)	93.9 (0.04)	93.9 (0.08)	93.9	92.50	0.04
	batched-EGNN	1.3 (1.0)	1.3 (0.1)	1.3 (0.12)	1.3 (0.14)	1.3	1.30	0.16
	EGNN-50	-	-	-	34.0 (1.0)	-	-	-
	SEED-GNN	2.0 (1.0)	0.8 (1.0)	2.3 (1.0)	2.3 (1.0)	5.4	2.2	1.0
GIN (86.11%)	FT-SEED-GNN	-0.8 (1.0)	-6.2 (1.0)	-4.5 (0.96)	-4.2 (0.96)	10.3	-4.10	0.94
	ENN-SEED-GNN	57.2 (1.0)	48.2 (0.2)	70.8 (0.24)	48.9 (0.22)	80.0	55.70	0.20
	batched-EGNN	-2.5 (1.0)	-2.5 (0.1)	-2.5 (0.16)	-2.5 (0.16)	-2.5	-2.50	0.18
	EGNN-50	-	-	-	30.8 (0.98)	-	-	-
	SEED-GNN	-1.7 (1.0)	-1.7 (0.9)	-4.2 (0.96)	-4.9 (1.0)	1.7	-3.60	0.96
	Coauthor-CS	18,333 Nodes	81,894 Edges	15 Classes	6,805 Features			
GCN (94.43%)	FT-SEED-GNN	0.6 (1.0)	0.8 (1.0)	0.9 (0.96)	2.5 (1.0)	3.5	1.30	0.99
	ENN-SEED-GNN	1.5 (1.0)	5.0 (1.0)	27.4 (0.92)	38.3 (0.98)	91.2	32.00	0.82
	batched-EGNN	-0.1 (1.0)	-0.1 (0.3)	-0.1 (0.4)	-0.1 (0.36)	-0.1	-0.10	0.39
	EGNN-50	-	-	-	13.1 (1.0)	-	-	-
	SEED-GNN	-0.1 (1.0)	-0.1 (1.0)	-0.2 (1.0)	-0.1 (1.0)	0.1	-0.0	1.0
Graph-SAGE (95.33%)	FT-SEED-GNN	0.6 (1.0)	0.8 (1.0)	0.9 (0.96)	2.5 (1.0)	3.5	1.30	0.99
	ENN-SEED-GNN	1.8 (1.0)	3.9 (0.8)	3.5 (0.8)	37.7 (0.72)	42.0	12.70	0.75
	batched-EGNN	0.0 (1.0)	0.0 (0.2)	0.0 (0.08)	0.0 (0.06)	0.0	0.00	0.15
	EGNN-50	-	-	-	17.8 (1.0)	-	-	-
	SEED-GNN	0.0 (1.0)	-0.1 (1.0)	0.2 (1.0)	0.7 (1.0)	1.5	0.5	1.0
GIN (91.60%)	FT-SEED-GNN	11.7 (0.0)	0.9 (0.9)	4.9 (0.92)	0.5 (1.0)	12.3	2.20	0.95
	ENN-SEED-GNN	25.2 (1.0)	84.3 (0.1)	84.0 (0.28)	76.6 (0.18)	89.1	74.70	0.23
	batched-EGNN	0.2 (1.0)	0.2 (0.1)	0.2 (0.08)	0.2 (0.06)	0.2	0.20	0.12
	EGNN-50	-	-	-	47.1 (1.0)	-	-	-
	SEED-GNN	-0.4 (1.0)	-1.0 (1.0)	-1.3 (1.0)	-1.1 (1.0)	-0.4	-1.2	0.99

D.2. SEED-GNN on YelpCHI

Last, we embrace the criticism that most datasets illustrated in Table 5 and Table 6 are not high-profile in nature for being rooted in shopping recommendation, citation connection, or something alike. Thus, we opt to evaluate SEED-GNN on the YelpCHI[3] (Rayana & Akoglu, 2015), which is a real-world collected dataset with the task being filtering out dishonest/fraudulent reviews. As failing to do so may impair the interests of business owners and customers, even leading to severe health and safety consequences due to the nature of hotels and restaurants, this dataset certainly carries a higher "stake" than the others. Table 16 shows SEED-GNN still performs well under this real-life scrutiny.

Table 16. GNN Model Editing Experiments on Real-world Graph. "DD" = Test Drawdown (smaller is better), "SR" = Success Rate ($\neq$ 1.0 implies insecure edits), "PE Acc." = Pre-edit Acc. Please refer to §3.2 for further metrics specifications.

Backbone (PE Acc.)	**Method**	**Test Drawdown (Success Rate)** 1th	10th	25th	50th	**Max DD**	**Avg DD**	**Avg SR**
	YelpChi	45,954 Nodes (14.5% Fraud)		3,846,979 Edges	3 Classes	32 Features		
GCN (89.80%)	FT	0.9 (1.0)	80.6 (1.0)	95.3 (1.0)	95.3 (1.0)	95.3	89.8	1.0
	ENN	3.9 (0.0)	-0.5 (0.3)	-0.5 (0.52)	-0.5 (0.56)	3.9	-0.4	0.46
	EGNN	36.4 (0.0)	80.9 (1.0)	89.6 (0.96)	93.3 (1.0)	93.3	84.2	0.94
	SEED-GNN	0.3 (0.0)	1.7 (1.0)	2.5 (0.96)	7.2 (0.96)	8.8	4.2	0.90
Graph-SAGE (86.60%)	FT	25.3 (1.0)	-4.5 (0.3)	70.1 (0.68)	15.0 (0.54)	76.7	24.3	0.49
	ENN	-8.8 (0.0)	-8.8 (0.3)	-8.8 (0.36)	-8.8 (0.48)	-8.8	-8.8	0.32
	EGNN	7.0 (1.0)	46.6 (0.5)	65.1 (0.88)	19.1 (0.42)	83.2	37.9	0.571
	SEED-GNN	-4.6 (0.0)	-2.9 (0.8)	5.5 (0.96)	14.2 (1.0)	14.2	3.2	0.91
GAT (87.6%)	FT	20.2 (1.0)	-14.5 (0.5)	-10.8 (0.48)	44.9 (0.62)	80.1	40.4	0.63
	ENN	82.9 (1.0)	82.9 (0.9)	82.9 (0.64)	82.9 (0.56)	82.9	82.9	0.70
	EGNN	-10.0 (1.0)	67.5 (0.5)	-5.8 (0.12)	6.5 (0.18)	81.5	24.1	0.38
	SEED-GNN	-10.0 (1.0)	-6.4 (0.9)	-5.1 (0.96)	-8.8 (1.0)	-0.2	-7.3	0.94
GIN (84.2%)	FT	1.7 (0.0)	16.9 (0.4)	34.2 (0.6)	23.7 (0.4)	36.2	15.7	0.37
	ENN	97.2 (1.0)	-2.4 (0.7)	-2.4 (0.68)	95.6 (0.32)	97.2	55.9	0.50
	EGNN	8.0 (1.0)	93.6 (0.9)	61.3 (0.52)	89.0 (0.82)	93.6	70.8	0.67
	SEED-GNN	-2.3 (0.0)	-0.3 (1.0)	5.8 (0.96)	8.0 (1.0)	14.3	5.0	0.93

[3] https://odds.cs.stonybrook.edu/yelpchi-dataset/