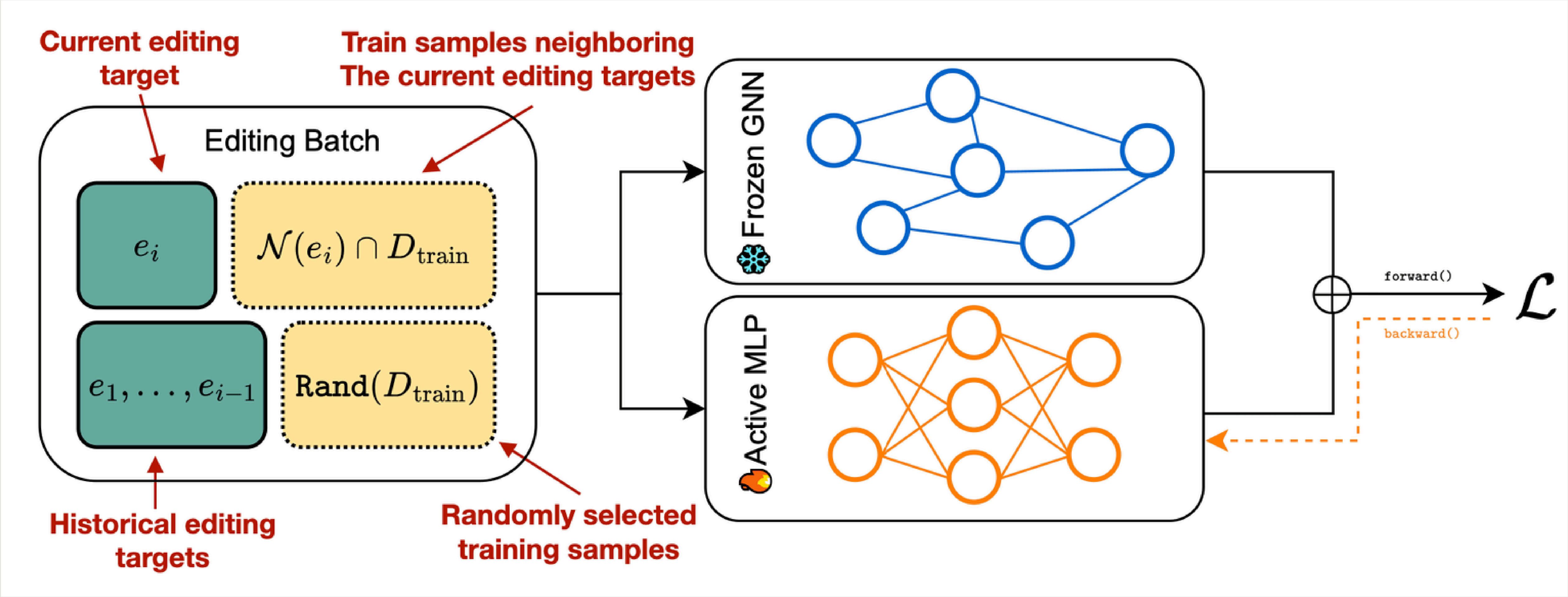
M.Alif Al Hakim

University of Indonesia
ui.ac.id

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

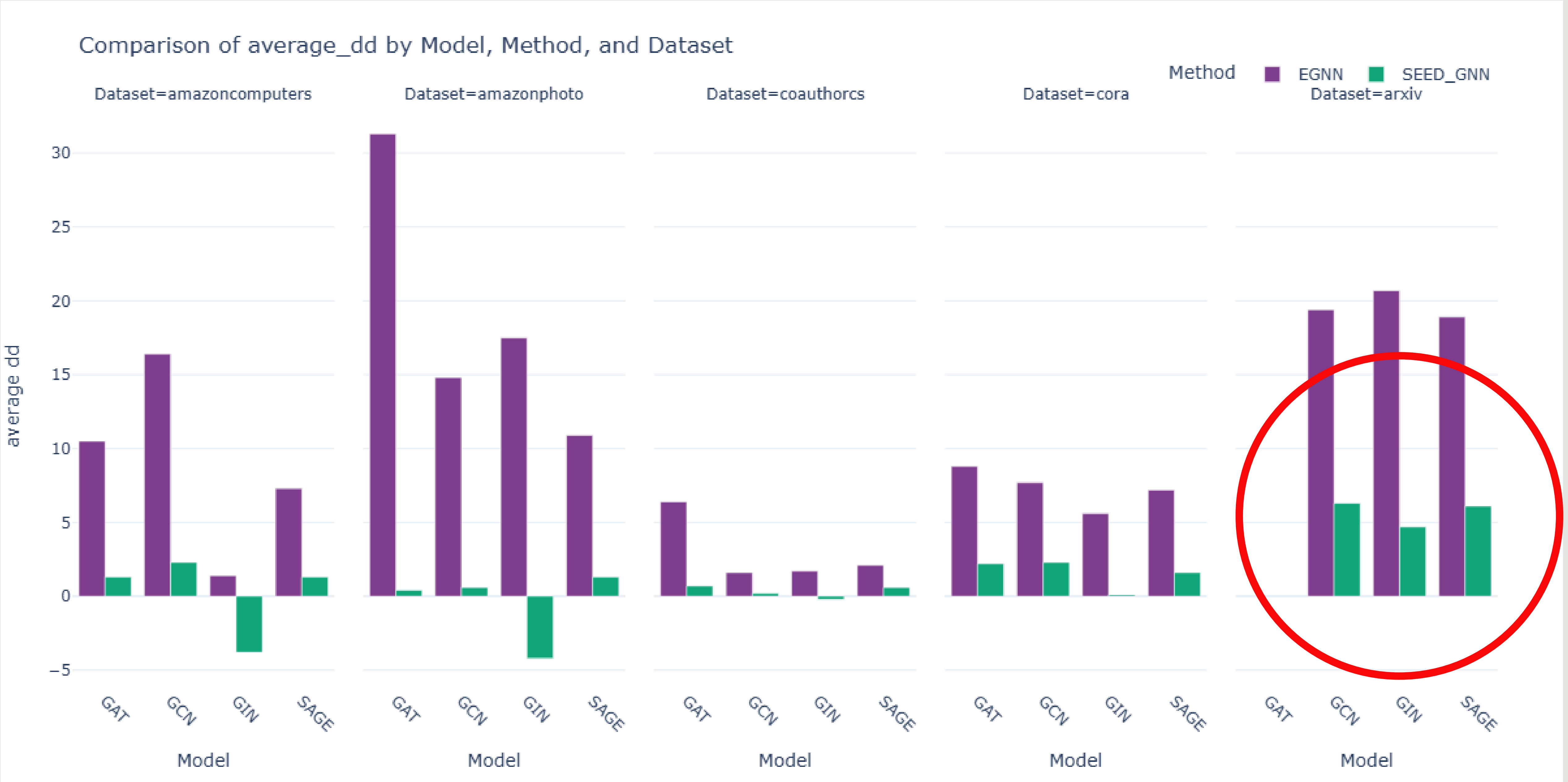
by Shaochen Zhong, Duy Le, Zirui Liu, et al.



Adapts **the EGNN (Editable Graph Neural Network for Node Classifications; Ziriu Liu et al.)** framework to edit a target e_i, constructing an editing batch with the **current** target, **previous edited** targets, its training-set **neighbors**, and **random** training samples

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

Reproduced Results



Original Experiment

	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
	ENN	48.2 (0.0)	48.2 (0.0)	48.2 (0.0)	48.2 (0.02)	48.2	48.2
	EGNN	3.0 (1.0)	9.2 (0.9)	17.9 (0.2)	4.2 (0.14)	58.4	13.0
	Adapter	48.7 (1.0)	70.0 (0.1)	64.3 (0.24)	68.4 (0.02)	70.0	60.2
	LoRA	3.9 (0.0)	3.9 (0.1)	3.9 (0.04)	3.9 (0.1)	3.9	3.9
	SEED-GNN	**-3.7 (1.0)**	**4.7 (1.0)**	**5.3 (1.0)**	**6.2 (1.0)**	**9.8**	**6.1**
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
	ENN	66.2 (0.0)	68.0 (0.0)	68.0 (0.04)	68.0 (0.06)	68.0	68.0
	EGNN	0.6 (1.0)	10.0 (0.5)	17.6 (0.32)	15.0 (0.36)	39.8	17.8
	Adapter	67.2 (1.0)	58.9 (0.2)	46.9 (0.32)	64.6 (0.14)	68.3	58.9
	LoRA	2.7 (0.0)	2.7 (0.0)	2.9 (0.2)	2.1 (0.1)	3.7	2.3
	SEED-GNN	**0.1 (1.0)**	**5.8 (1.0)**	**5.5 (1.0)**	**4.8 (1.0)**	**11.0**	**4.9**
GIN (66.17%)	FT	61.9 (1.0)	46.0 (0.2)	63.6 (0.16)	63.7 (0.02)	65.9	59.9
	ENN	64.9 (0.0)	41.8 (0.1)	44.9 (0.16)	44.4 (0.14)	66.1	51.6
	EGNN	**0.2 (1.0)**	34.6 (0.6)	22.4 (0.32)	30.0 (0.22)	53.0	17.0
	Adapter	65.3 (1.0)	57.9 (0.2)	63.6 (0.12)	60.1 (0.12)	65.3	54.6
	SEED-GNN	0.7 (1.0)	**1.1 (0.8)**	**4.9 (0.96)**	**6.7 (0.98)**	**8.6**	**5.2**

Max DD **Avg DD**

On Large-Scale Graphs: **Average Drawdown Exceeds 5%**

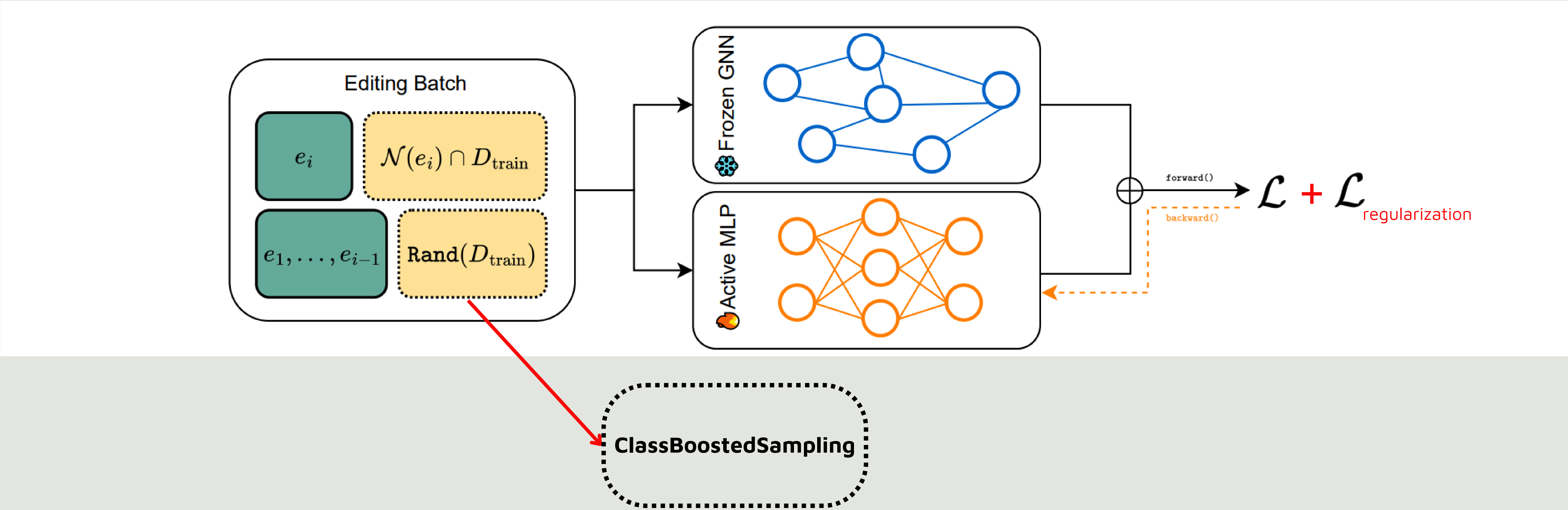
Full Experiment Results: https://github.com/malifalhakim/gnn_editing/tree/main/exp_result/tables

Notes: GAT encountered an Out Of Memory (OOM) error during prediction on the ArXiv dataset due to computational memory limitations.

What If?

Since the paper says "**The Enemy is, Once Again, Overfitting**", Why don't we try to improve method's generalization capability. In this experiment, I added:

- **L2 regularization within the MLP model.**
- **Class-Boosted Stratified Sampling for training data, replacing the previous sampling method Rand(D_train)**, will increase the representation of historically mispredicted classes while attempting to maintain the original distribution.



What If?

Regularizations

$$\mathcal{L}_{\text{reg}} = \underbrace{\lambda_{\text{act}} \sum_{l=1}^{L-1} \|\mathbf{H}^{(l)}\|_F^2}_{\text{Activity Regularization}} + \underbrace{\lambda_w \sum_{k=1}^{K} \|\mathbf{W}^{(k)}\|_F^2}_{\text{Weight Decay}}$$

```python
class MLP(BaseModel):
    def forward(self, x: Tensor, *args, **kwargs) -> Tensor:
            **kwargs: Additional keyword arguments

        Returns:
            Output tensor with shape [batch_size, out_channels]
        """
        self.activity_reg_loss = 0.0

        for idx in range(self.num_layers - 1):
            lin = self.lins[idx]
            h = lin(x, *args, **kwargs)
            if self.batch_norm:
                h = self.bns[idx](h)
            if self.layer_norm:
                h = self.lns[idx](h)
            if self.residual and h.size(-1) == x.size(-1):
                min_size = min(h.size(0), x.size(0))
                h[:min_size] += x[:min_size]

            # Add activity regularization (L2 penalty on activations)
            if self.activity_regularization > 0:
                self.activity_reg_loss += self.activity_regularization * torch.sum(h**2)

            x = self.activation(h)
```

Class Boosted Stratified Sampling

$$p_c^{\text{boosted}} = \frac{p_c}{\beta \cdot p_{c^*} + \sum_{c \neq c^*} p_c}$$

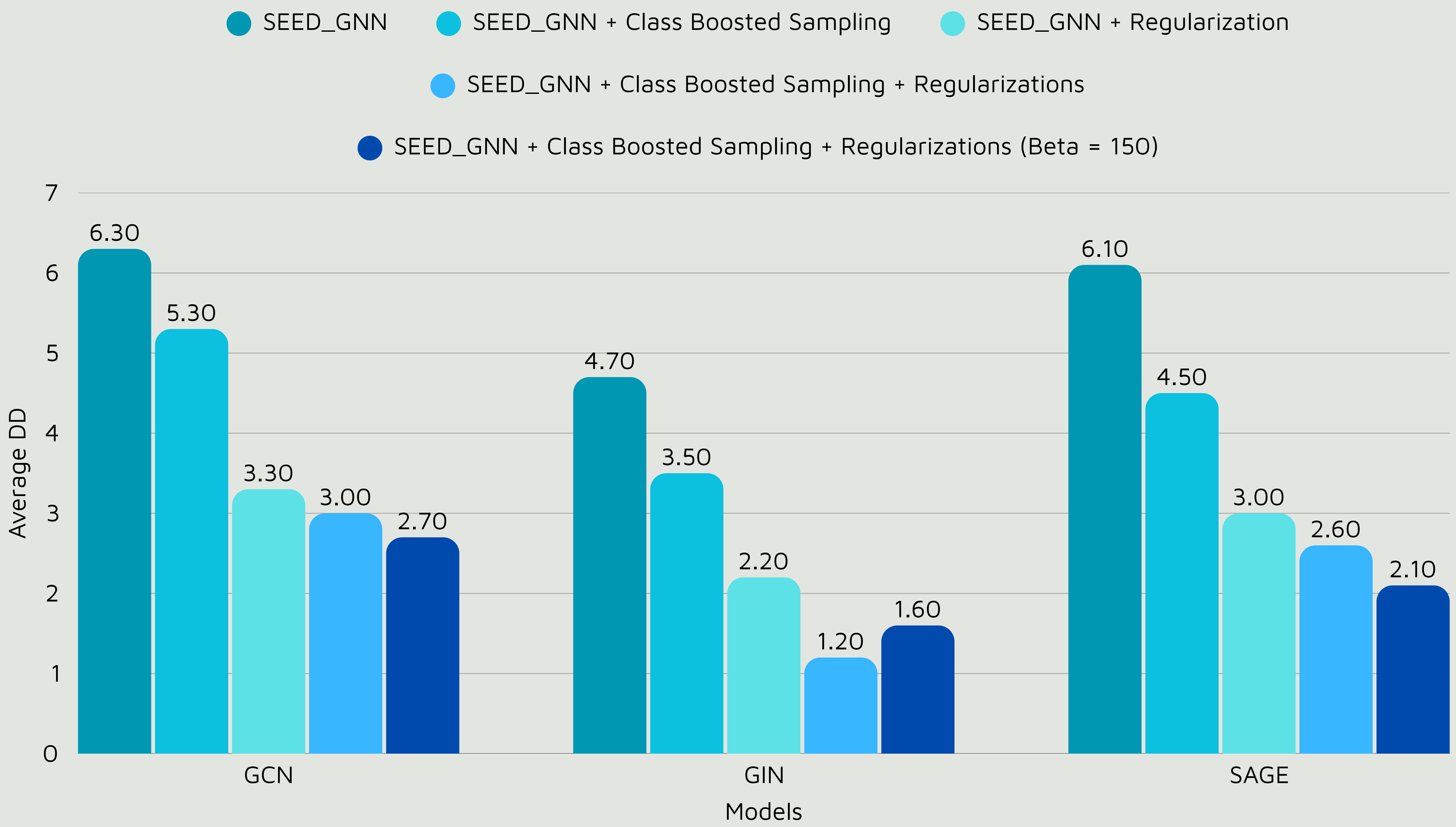
$$p_{c^*}^{\text{boosted}} = \frac{\beta \cdot p_{c^*}}{\beta \cdot p_{c^*} + \sum_{c \neq c^*} p_c}$$

$p_c = \frac{\text{count}(c)}{|D_{\text{train}}|}$: Original proportion of class c in the training

c^*: Edit target class (the class of e_i)

```python
def _select_mixup_training_nodes(
    # Calculate original class distribution in training set
    train_y = whole_data.y[whole_data.train_mask]
    class_counts = [(train_y == c).sum().item() for c in range(num_classes)]
    total_train_nodes = len(train_y)
    class_proportions = [count / total_train_nodes for count in class_counts]

    # Calculate nodes per class based on original distribution with a boost factor for the target class
    boost_factor = 1.2
    target_boost = min(boost_factor, 1.0 / class_proportions[target_class])  # Prevent over-allocation

    # Normalize proportions after boosting target class
    adjusted_proportions = class_proportions.copy()
    adjusted_proportions[target_class] *= target_boost
    total_adjusted = sum(adjusted_proportions)
    adjusted_proportions = [p / total_adjusted for p in adjusted_proportions]

    # Calculate nodes per class based on adjusted proportions
    nodes_per_class = {
        c: max(1, int(num_general_nodes * adjusted_proportions[c]))
        for c in range(num_classes)
    }

    # Ensure we don't exceed the target number of nodes
    total_allocated = sum(nodes_per_class.values())
    if total_allocated > num_general_nodes:
        # Scale down proportionally
        scale_factor = num_general_nodes / total_allocated
        nodes_per_class = {c: max(1, int(n * scale_factor)) for c, n in nodes_per_class.items()}
```

Empirical Results

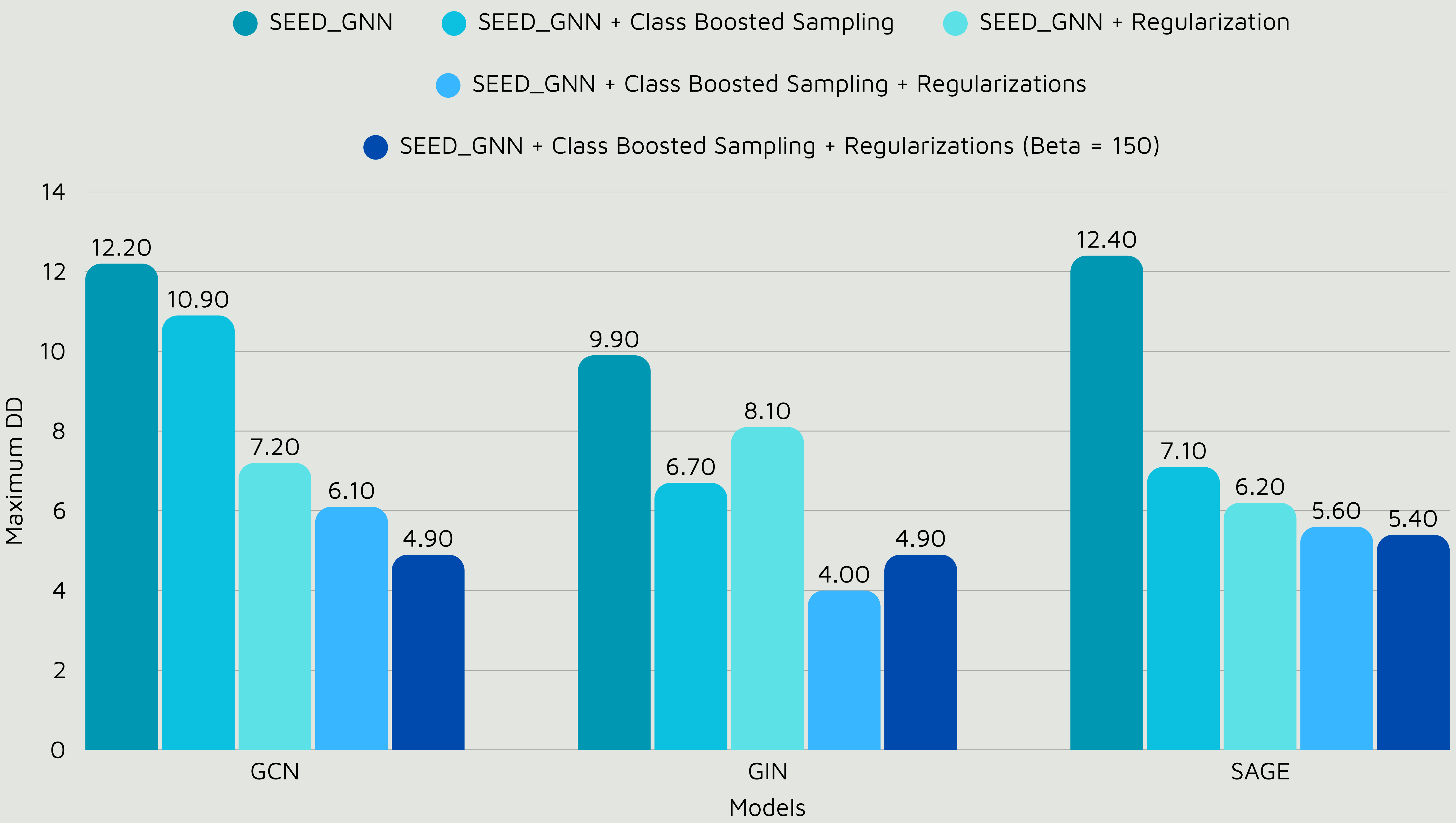


SEED-GNN with Class-Boosted Sampling and Regularization

Achieves **Up to 3.9 × Lower Average DD** While Maintaining >98% Edit Success Rate

Notes: Result on ArXiv Dataset using default parameter

Empirical Results

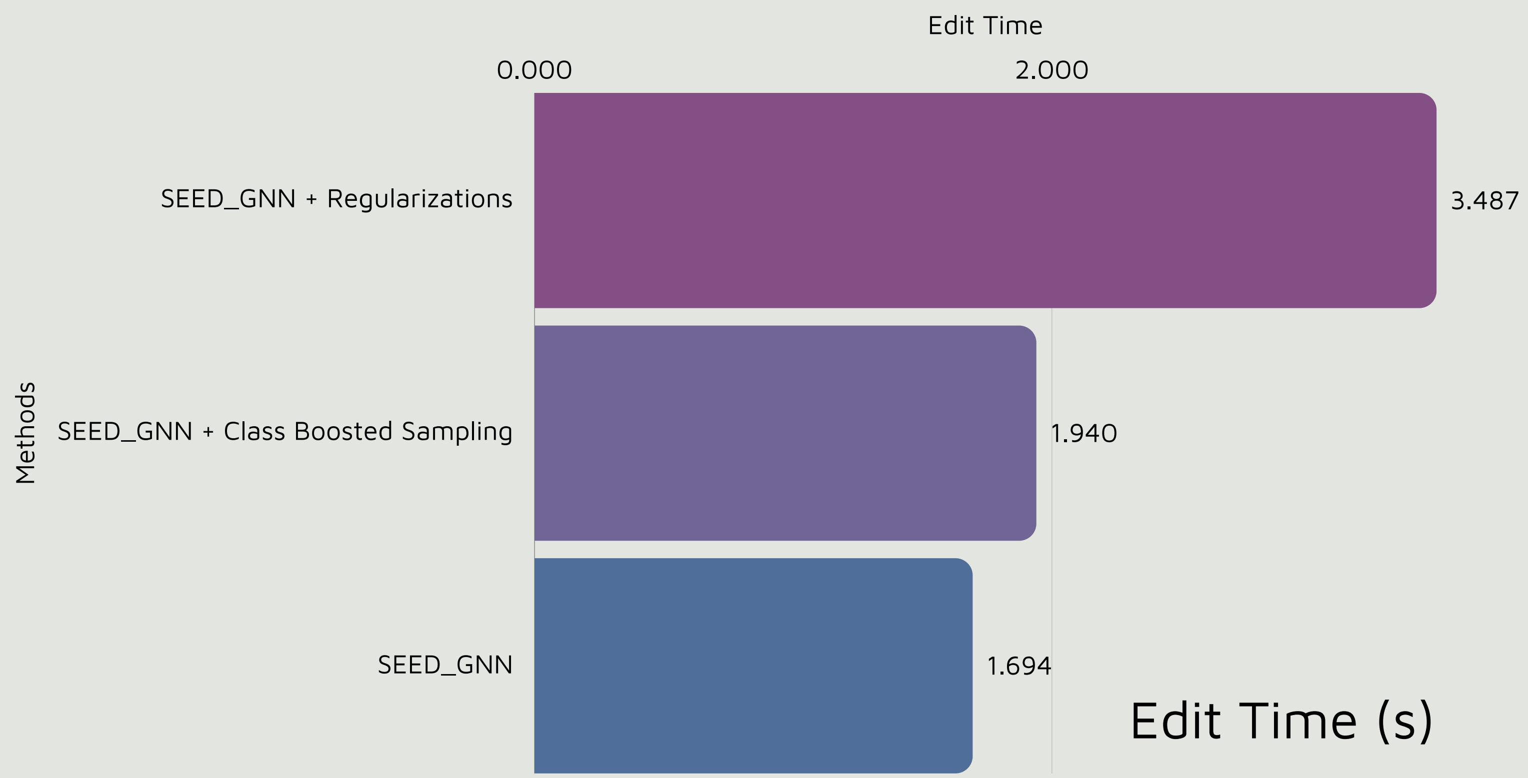


Achieves up to **2.4x lower Maximum Drawdown (Max DD)** while also **reducing the difference** between Maximum and Average Drawdown **to 3.5% or less**

Notes: Result on ArXiv Dataset using default parameter

Tradeoff

Slower Editing Time



Regularizations Introduce Computational Trade-offs: SEED-GNN with Regularizations **Doubles Edit Time** (106% Increase) to **Enforce Model Constraints**, While Class-Boosted Sampling **Adds Minimal Overhead (14%)**.

Notes: Result on ArXiv Dataset using GCN model

While this trade-off has potential benefits, the experiment indicates a **possible decrease in the success rate**, with an observed decrement of **roughly 1 percent on average.**

Conclusion

- SEED-GNN provides a method for handling sequential editing of GNN models, improving upon the previous EGGN method, which struggled to maintain performance in sequential settings
- SEED-GNN primarily prevents overfitting in the active MLP model by modifying the Edit Batch
- Adding more constraints to the MLP model could improve its generalization, but this will involve trade-offs.