Results (Updated)

Datasets	GLADC	
MMP	**0.696 ± 0.042**	→ **0.508 ± 0.138**
HSE	**0.618 ± 0.110**	→ **0.551 ± 0.070**
p53	**0.649 ± 0.216**	→ **0.497 ± 0.122**
BZR	**0.715 ± 0.067**	→ **0.683 ± 0.045**
DHFR	**0.612 ± 0.041**	→ **0.560 ± 0.053**
COX2	**0.615 ± 0.044**	→ **0.595 ± 0.093**
ENZYMES	**0.583 ± 0.035**	→ **0.529 ± 0.071**
IMDB	**0.656 ± 0.023**	→
AIDS	0.993 ± 0.005	→ **0.993 ± 0.005**
NCI1	0.683 ± 0.011	→ **0.330 ± 0.016**

Results (Updated) cont.

Attributed datasets tested with plain graph procedure

- Not accounting for dataset node features
- Computing node degree either way
- Missing parameter specification

```
parser.add_argument('--feature', dest='feature', default='deg-num', help='use what node feature')
```

Plain Graphs

```python
elif features == 'deg-num':
    degs = np.sum(np.array(adj), 1)
    if self.max_num_nodes > G.number_of_nodes():
        degs = np.expand_dims(np.pad(degs, (0, self.max_num_nodes - G.number_of_nodes()), 'constant', constant_values=0),
                              axis=1)
    elif self.max_num_nodes < G.number_of_nodes():
        deg_index = np.argsort(degs, axis=0)
        deg_ind = deg_index[0: G.number_of_nodes()-self.max_num_nodes]
        degs = np.delete(degs, [deg_ind], axis=0)
        degs = np.expand_dims(degs, axis=1)
    else:
        degs = np.expand_dims(degs, axis=1)
    self.feature_all.append(degs)
```

- Compute ‘degs’ (1D array) as the sum of each row of ‘adj’ (square matrix)
    - ‘deg[i]’ represents degree of node i
    - Degree is the number of edges connected to a node
- If graph has more nodes than expected (‘max_num_nodes’), we remove nodes with least amount of edges
    - Feature dimension consistency across all graphs
    - If less than ‘max_num_nodes’, we add padding

Loss

As formulated in the paper

$$L_{total} = L_1 + L_2 + L_3.$$

And expanded…

$$L_1 = \left\|\mathbf{A} - \hat{\mathbf{A}}\right\|_F^2 + \left\|\mathbf{X} - \hat{\mathbf{X}}\right\|_F^2.$$

$$L_2 = -log \frac{exp\left(sim\left(\hat{\mathbf{Z}}_{\acute{G}i}, \mathbf{Z}_{\acute{G}i}\right)/\tau\right)}{\sum_{\acute{i}=1,\acute{i}\neq i}^{N} exp\left(sim\left(\mathbf{Z}_{\acute{G}i}, \mathbf{Z}_{\acute{G}\acute{i}}\right)/\tau\right)},$$

$$L_3 = L_{node} + L_{graph}.$$

Loss (cont.)

- In code:
    - ```
      lossG = err_g_con_s + err_g_con_x + graph_loss + node_loss + err_g_enc
      ```
- Where:
    - L1:
        - ‘err_g_con_s’ + ‘err_g_con_x’
    - L2:
        - ‘err_g_enc’
    - L3:
        - ‘graph_loss’ + ‘node_loss’
- *Note L2 carries no node contrastive learning paradigm*
      ```

Loss (cont.)

Some notation:

- **h0** -> real node features
- **adj_label** -> real adjacency matrix
- **x1_r** -> node-level latent feature representation (array where each row corresponds to a node feature)
- **Feat_0** -> graph-level latent representation
- **x1_r_1** -> randomized node-level latent representation
- **Feat_0_1** -> randomized graph-level latent representation
- **x_fake** -> reconstructed node features
- **s_fake** -> reconstructed adjacency matrix
- **x2** -> node-level latent feature representation of reconstructed feature array
- **Feat_1** -> graph-level latent representation of reconstructed adjacency matrix

Loss (cont.)

- L1:
  - ```
    err_g_con_s, err_g_con_x = loss_func(adj_label, s_fake, h0, x_fake)
    ```
- L2:
  - ```
    err_g_enc=loss_cal(Feat_0_1, Feat_0)
    ```
- L3:
  - ```
    node_loss=torch.mean(F.mse_loss(x1_r, x2, reduction='none'), dim=2).mean(dim=1).mean(dim=0)
    ```
  - ```
    graph_loss = F.mse_loss(Feat_0, Feat_1, reduction='none').mean(dim=1).mean(dim=0)
    ```

L1

- ```
  err_g_con_s, err_g_con_x = loss_func(adj_label, s_fake, h0, x_fake)
  ```
    - Loss to measure how well the reconstruction matches the original
    - Steps:
        - Squared differences (s_fake - adj_label)^2 and (x_fake - h0)^2
        - Summations of those differences (along dimension 1)
        - Square root of those square differences
    - Basically a form of **Euclidean distance** loss for both node features and graph structure
  ```

L2

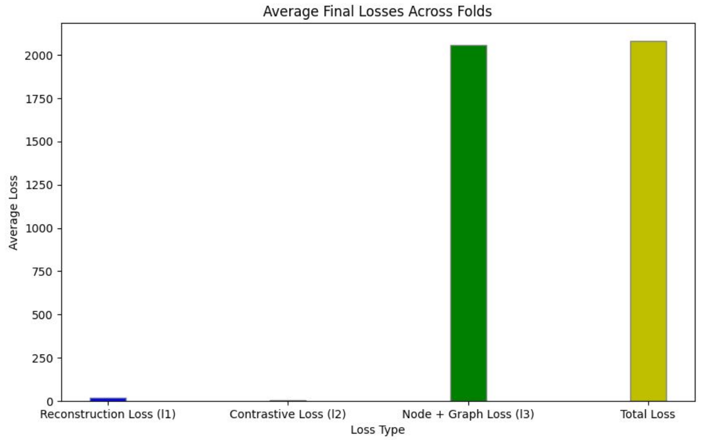
- ```
  err_g_enc=loss_cal(Feat_0_1, Feat_0)
  ```
  - Contrastive loss (ensures that model can distinguish between different views of the graph)
  - ‘loss_cal()’ follows formula proposed in paper
  ```

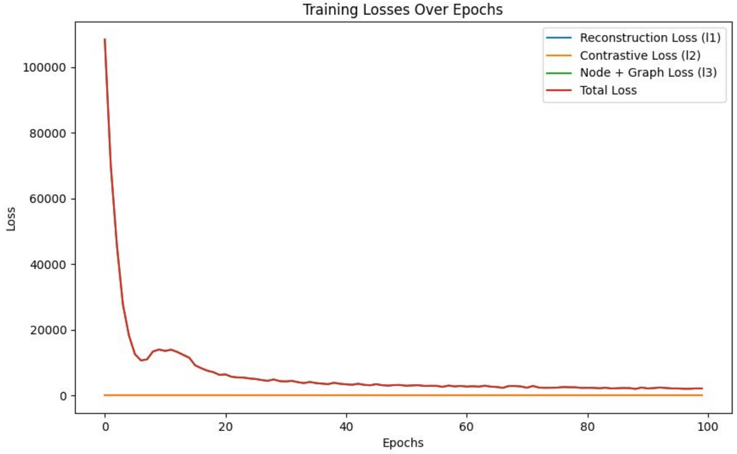
L3

- ```
  node_loss=torch.mean(F.mse_loss(x1_r, x2, reduction='none'), dim=2).mean(dim=1).mean(dim=0)
  ```
    - Mean squared error of latent node-level feature representations
- ```
  graph_loss = F.mse_loss(Feat_0, Feat_1, reduction='none').mean(dim=1).mean(dim=0)
  ```
    - Mean squared error of latent adjacency matrix representations

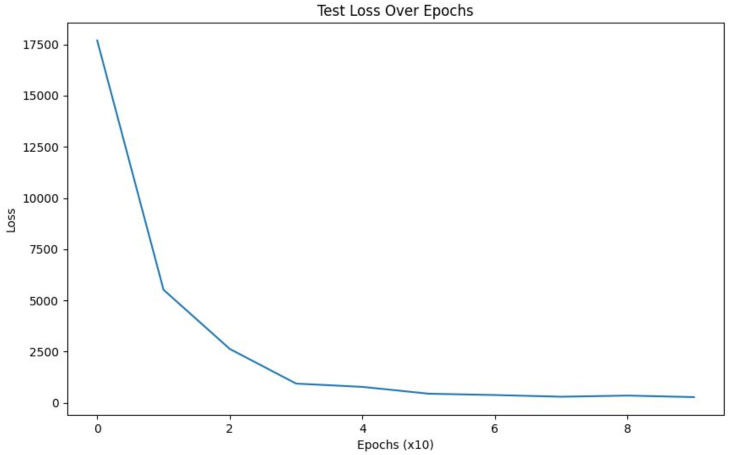
Loss Evolution (BZR)

- Averaged across 5 folds or trials



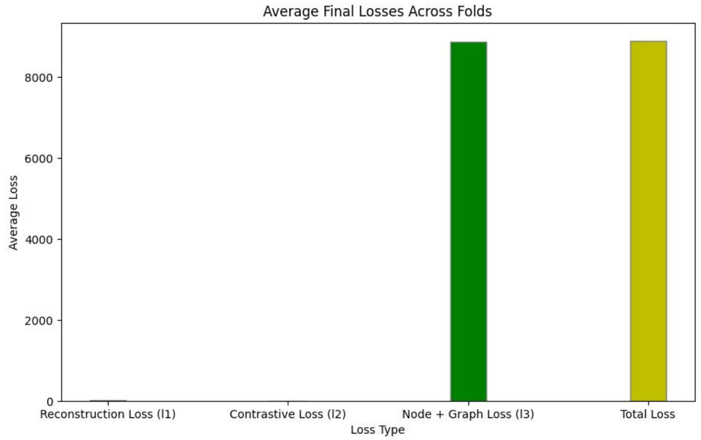


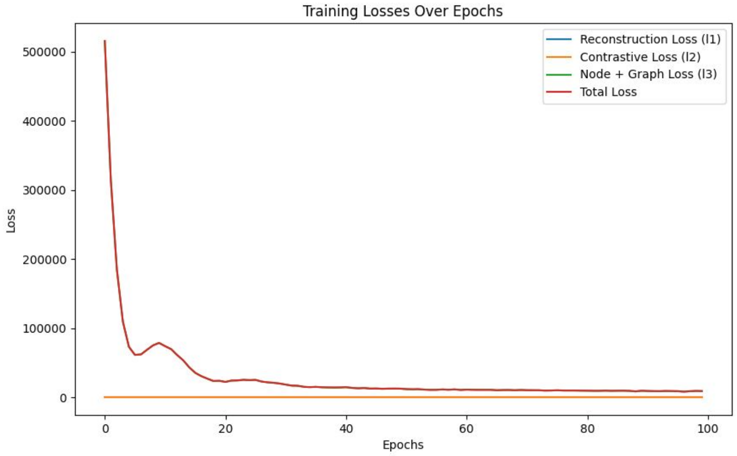
Test loss computed only from L3 for every DS

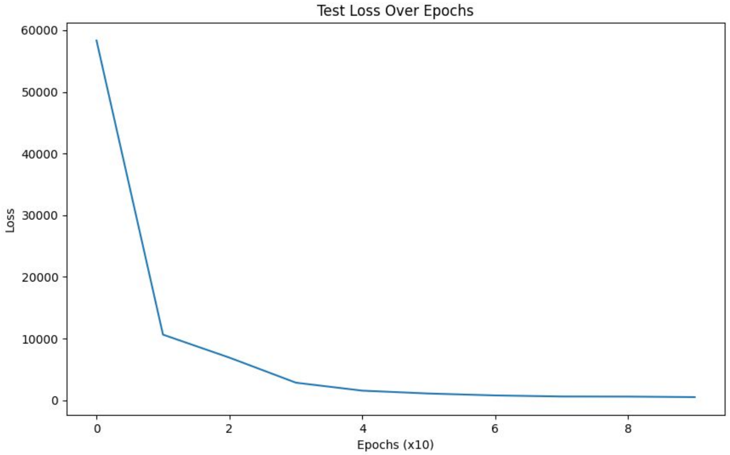


Loss Evolution (DHFR)

- Averaged across 5 folds or trials

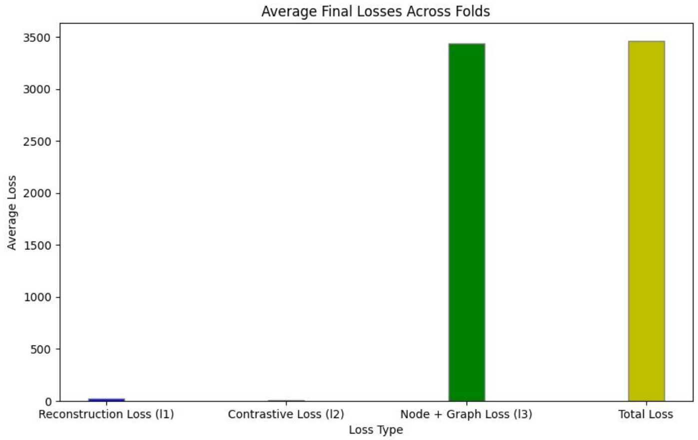


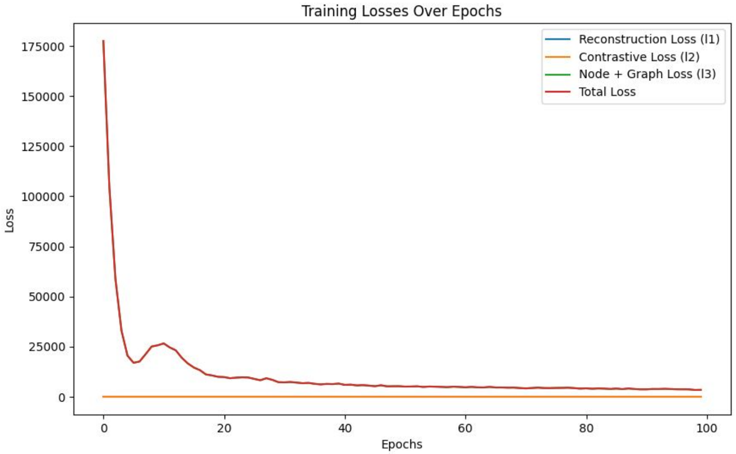


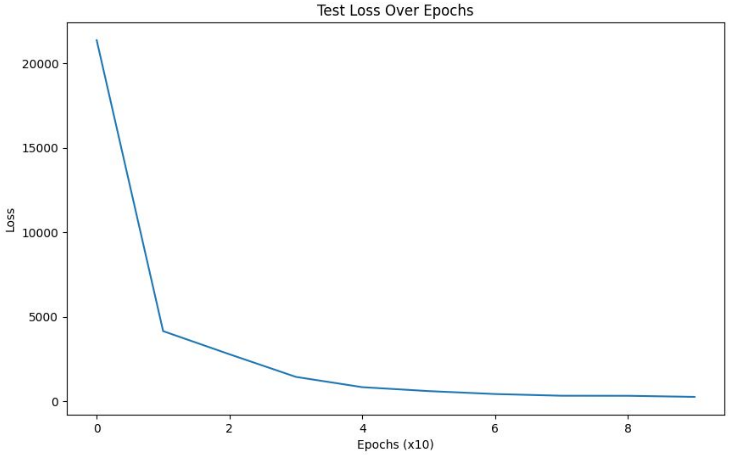


Loss Evolution (COX2)

- Averaged across 5 folds or trials

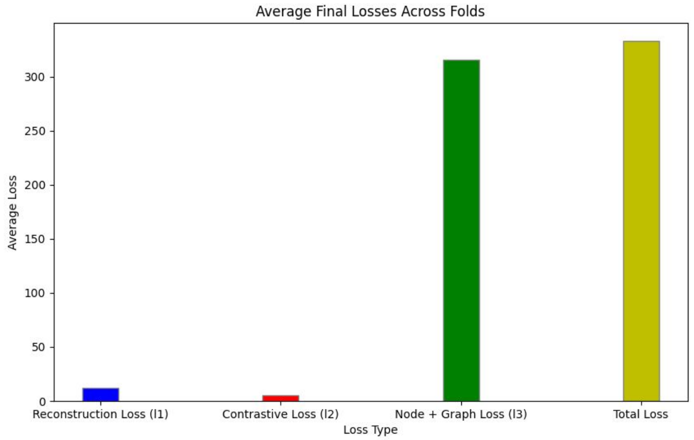


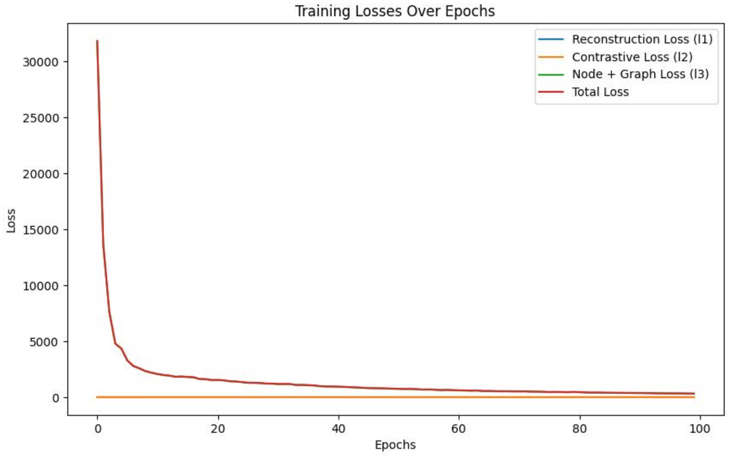


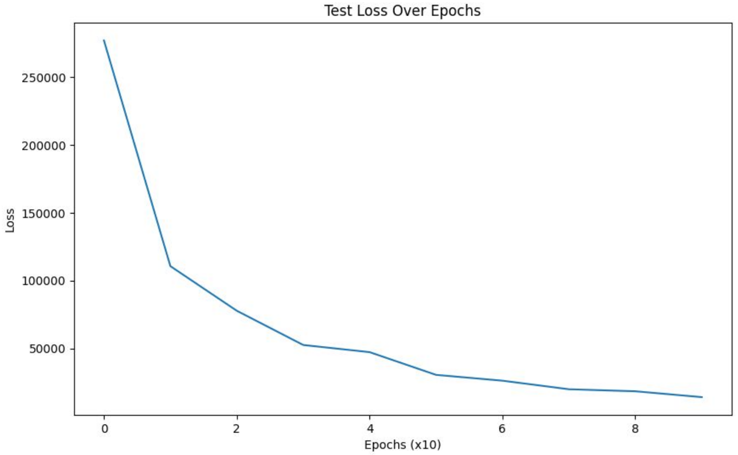


Loss Evolution (AIDS)

- Averaged across 5 folds or trials

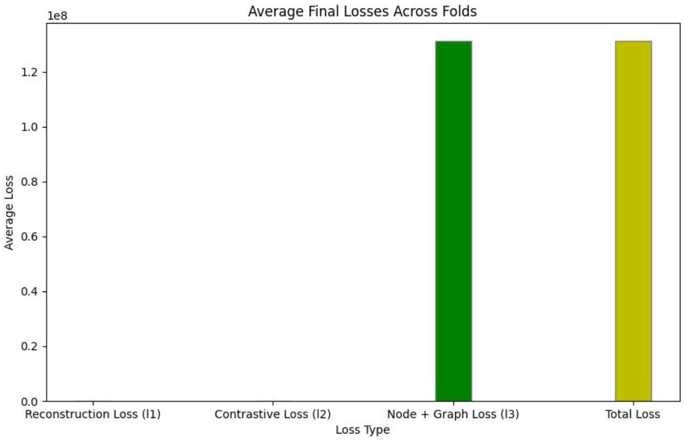


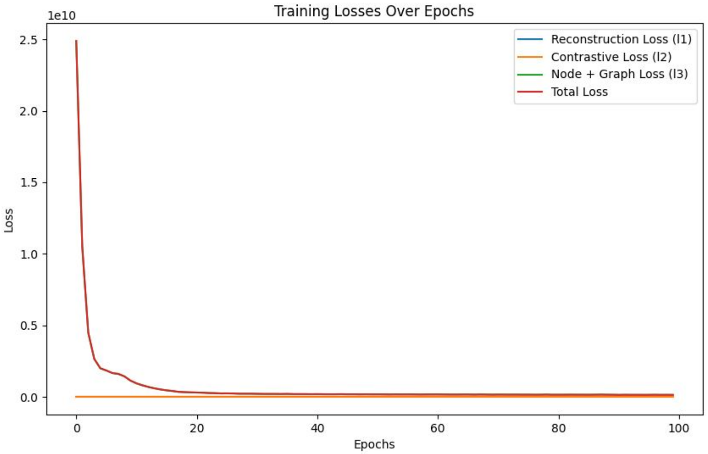


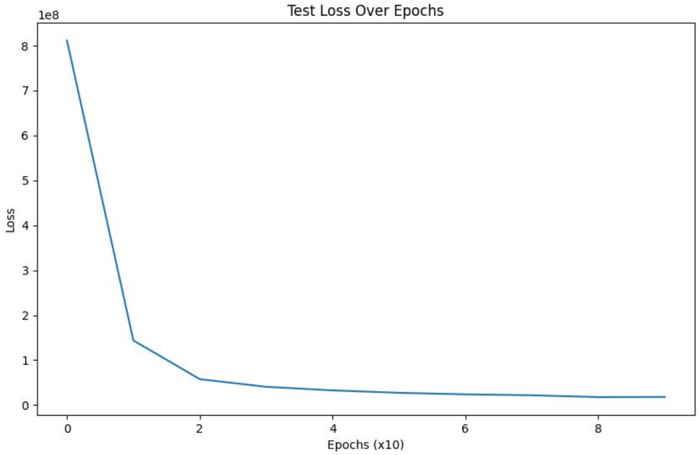


Loss Evolution (ENZYMES)

- Averaged across 5 folds or trials

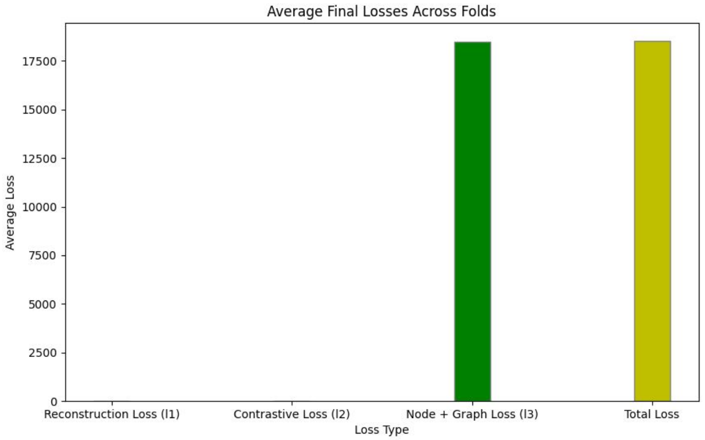


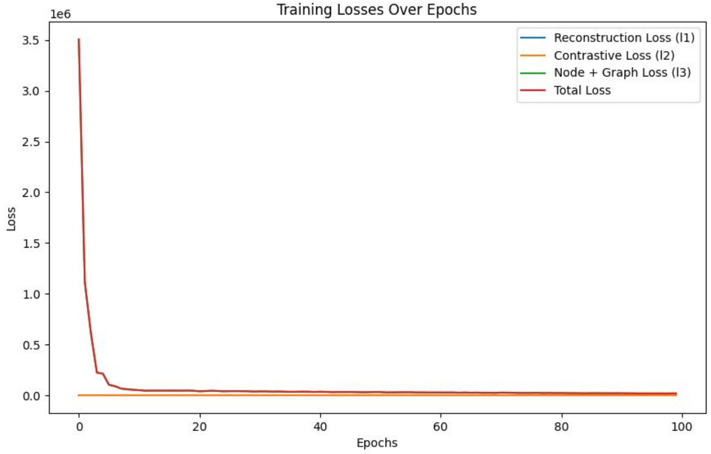


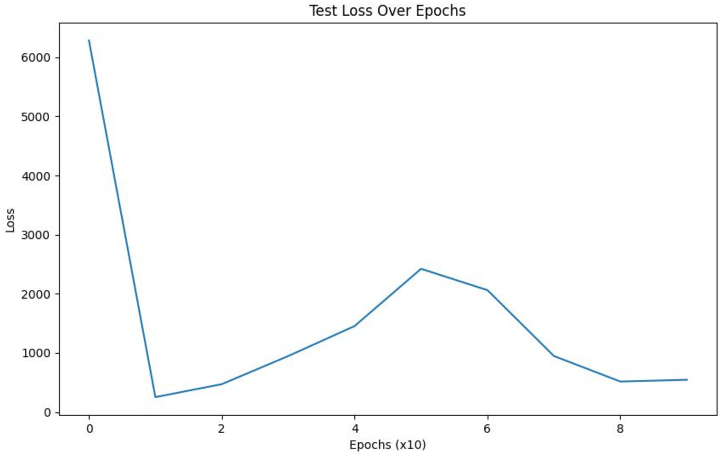


Loss Evolution (NCI1)

- Averaged across 5 folds or trials

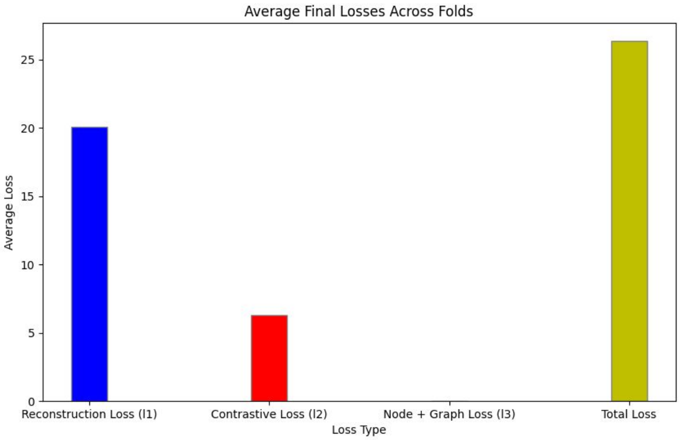


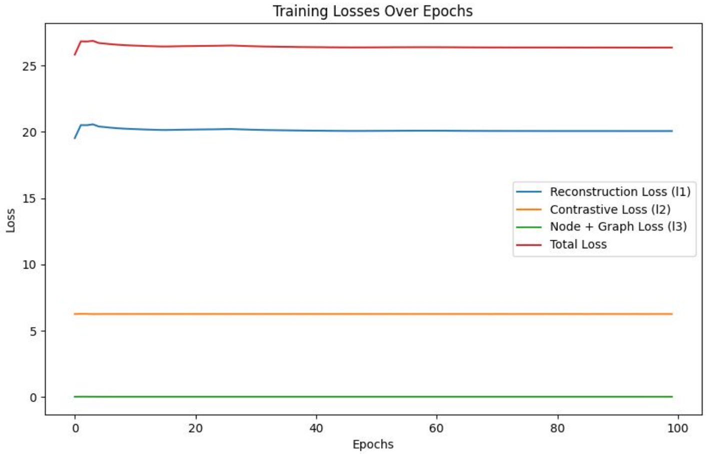


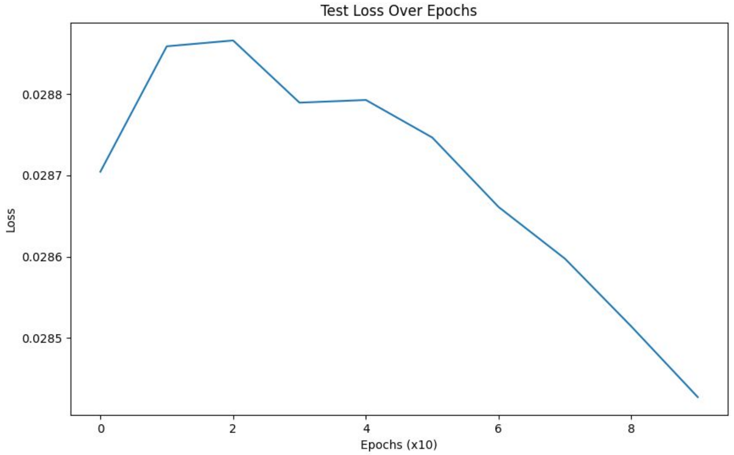


Loss Evolution (p53)

- Averaged across 5 folds or trials

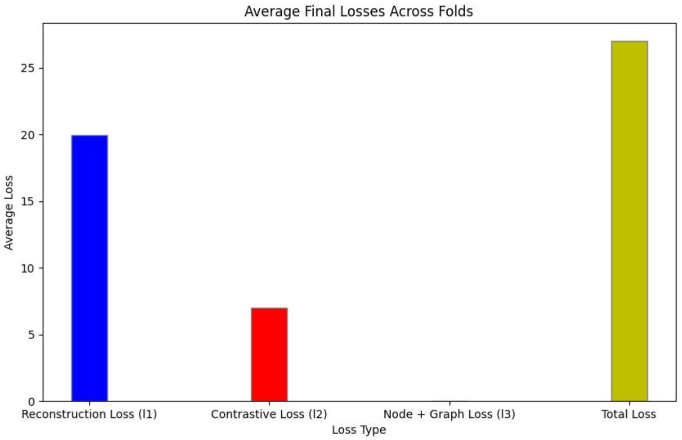


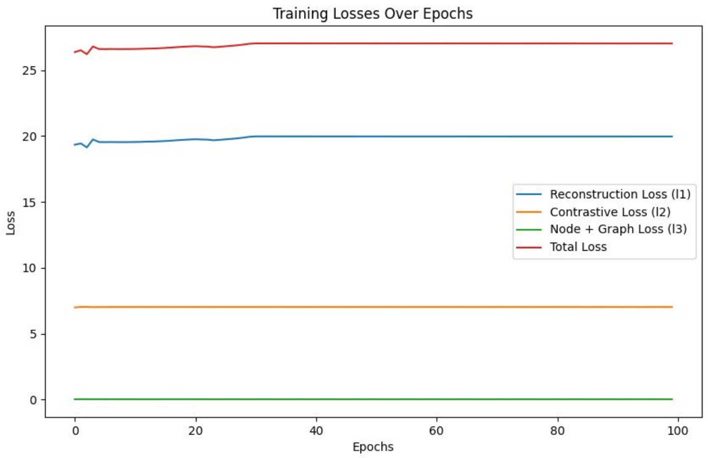


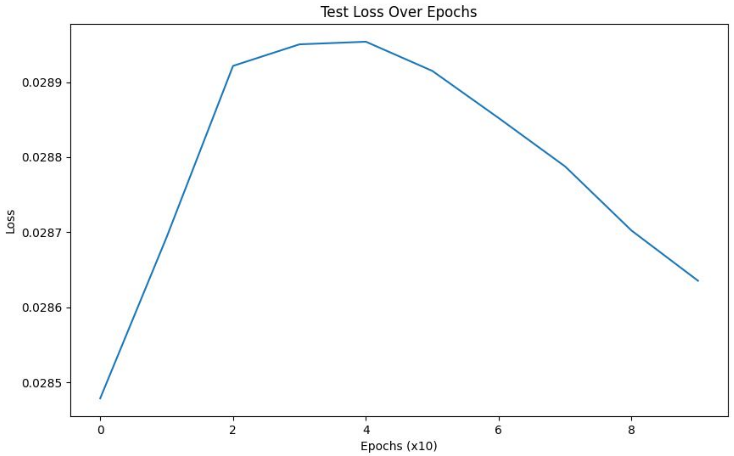


Loss Evolution (MMP)

- Averaged across 5 folds or trials

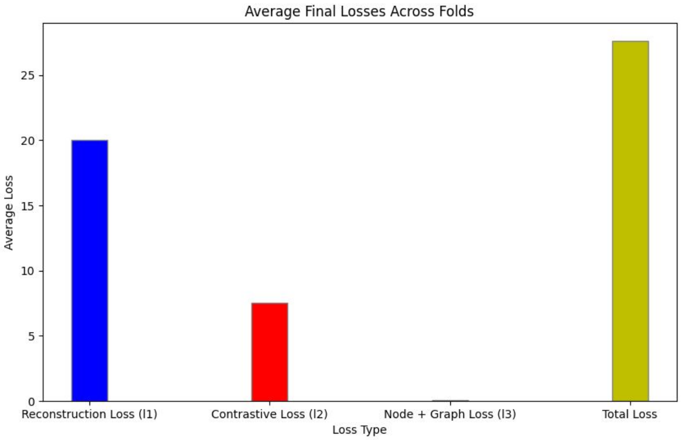


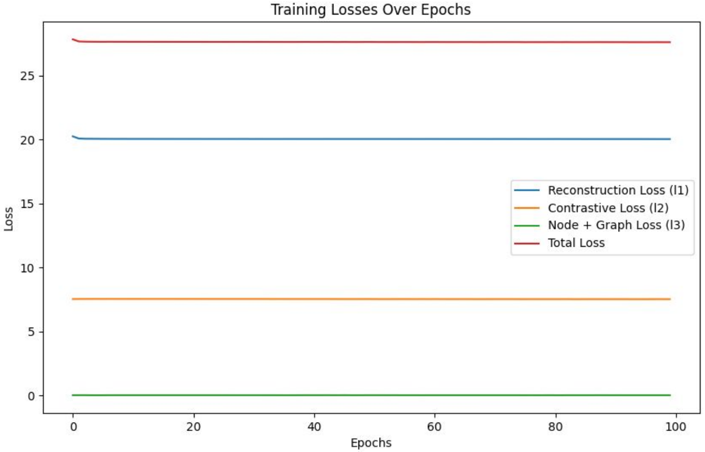


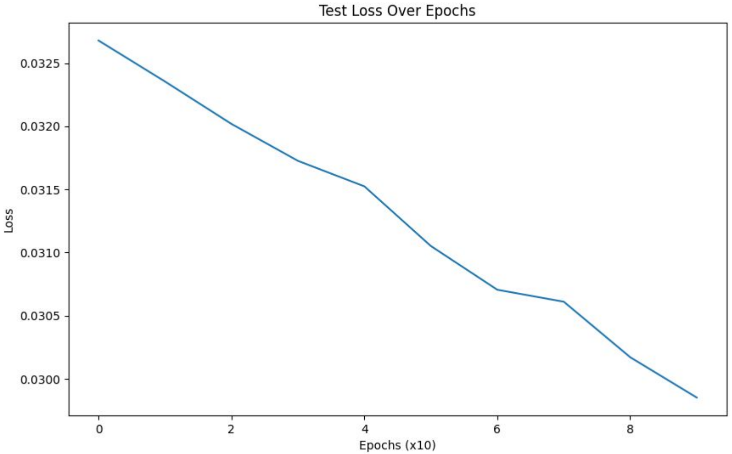


Loss Evolution (HSE)

- Averaged across 5 folds or trials







Discussion

- Big disparity on l3 loss between datasets trained on linear layers vs graph convolution layers.
    - L3 in linear layers model is extremely higher in magnitude than the other 2 losses.
    - L3 in graph convolution layers model is negligible compared to the other two.

Future Steps

- Try different configuration of the loss function (ie. exclude L3)?
- Try running attributed datasets through gc layers (refactor ‘main.py’)?