High-throughput Screening of Promising Redox-Active Molecules with MolGAT

Mesfin Diro Chaka [a,c,*], Chernet Amente Geffe [a], Alex Rodriguez [d], Nicola Seriani [d], Qin Wu [e], Yedilfana Setarge Mekonnen [b,*]

[a] *Department of Physics, College of Natural and Computational Sciences, Addis Ababa University, P. O. Box 1176, Addis Ababa, Ethiopia,*

[b] *Center for Environmental Science, College of Natural and Computational Sciences, Addis Ababa University, P. O. Box 1176, Addis Ababa, Ethiopia,*

[c] *Computational Data Science, College of Natural and Computational Sciences, Addis Ababa University, P. O. Box 1176, Addis Ababa, Ethiopia,*

[d] *The Abdus Salam International Centre for Theoretical Physics(ICTP),, Condensed Matter and Statistical Physics Section, 34100, Trieste, Italy,*

[e] *Brookhaven National Laboratory, Center for Functional Nanomaterials, Upton,, NY, US,*

[*] Corresponding author

Email address: yedilfana.setarge@aau.edu.et (Yedilfana Setarge Mekonnen)

June 3, 2023

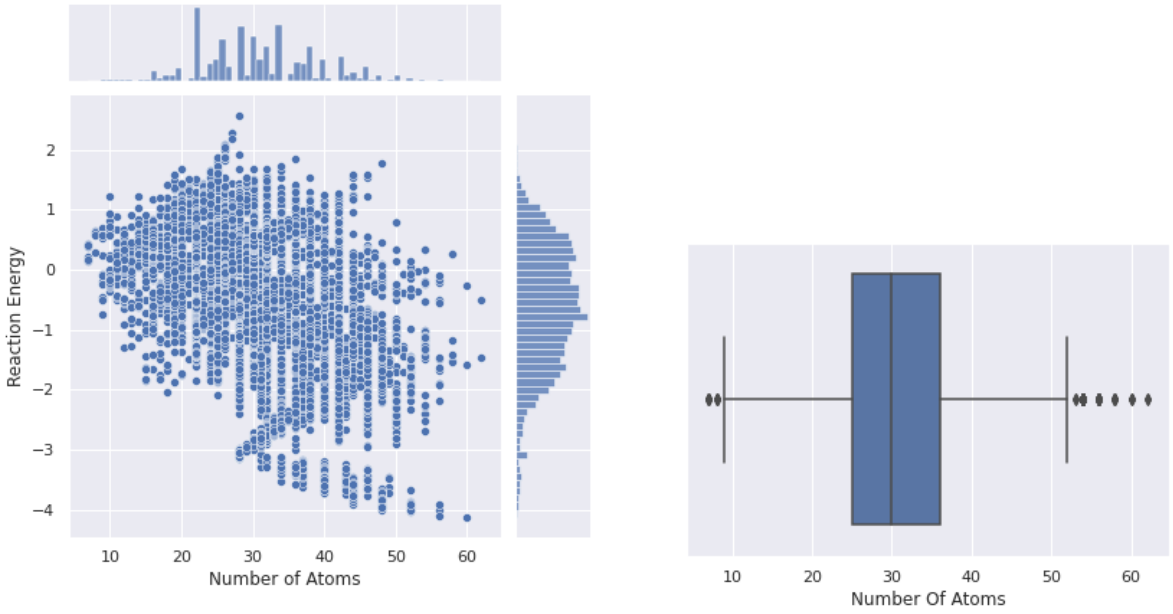
1. RedDB Dataset

Table S 1: Sample smiles and reaction energy in eV from RedDB dataset used for training

	Smiles	Reaction energy(eV)
0	O=C1CC(=O)C=C1C(=O)O	-0.115667
1	O=C1CC(=O)C=C1	-0.496880
2	O=C1CC(=O)C(C(=O)O)=C1C(=O)O	-1.310897
3	O=C1CC(=O)C=C1F	-0.446811
4	O=C1CC(=O)C(F)=C1F	-0.441369
5	O=C1CC(=O)C(N)=C1N	0.784777
6	O=C1CC(=O)C=C1N	0.344224
7	O=C1CC(=O)C=C1O	-0.073199
8	O=C1CC(=O)C(O)=C1O	0.201364
9	O=C1CC(=O)C(=C1)S(=O)(=O)O	-0.442741
10	O=C1CC(=O)C(S(=O)(=O)O)=C1S(=O)(=O)O	-0.866443
11	O=C1NC(=O)C(=C1)C(=O)O	-0.886275
12	O=C1NC(=O)C(C(=O)O)=C1C(=O)O	-1.661161
13	O=C1NC(=O)C=C1	-0.198915
14	O=C1NC(=O)C(F)=C1F	-0.032109
15	O=C1NC(=O)C=C1F	-0.082451
16	O=C1NC(=O)C(=C1)N	0.576337
17	O=C1NC(=O)C(N)=C1N	0.655795
18	O=C1NC(=O)C(O)=C1O	0.465315
19	O=C1NC(=O)C(=C1)O	0.252794
20	O=C1NC(=O)C(S(=O)(=O)O)=C1S(=O)(=O)O	-0.593772
21	O=C1NC(=O)C(=C1)S(=O)(=O)O	-0.729942
22	O=C1SC(=O)C=C1	-0.506404
23	O=C1SC(=O)C(=C1)C(=O)O	-0.903691
24	O=C1SC(=O)C(C(=O)O)=C1C(=O)O	-1.718570
25	O=C1SC(=O)C(F)=C1	-0.453070
26	O=C1SC(=O)C(F)=C1F	-0.454975
27	O=C1SC(=O)C(=C1)N	0.312659
28	O=C1SC(=O)C(N)=C1N	0.314564
29	O=C1SC(=O)C(O)=C1O	0.102315
30	O=C1SC(=O)C(=C1)O	-0.075376
31	O=C1SC(=O)C(S(=O)(=O)O)=C1S(=O)(=O)O	-0.821259
32	O=C1SC(=O)C(=C1)S(=O)(=O)O	-0.950068
33	O=C1OC(=O)C(C(=O)O)=C1C(=O)O	-1.417782
34	O=C1OC(=O)C=C1	-0.192929
35	O=C1OC(=O)C(=C1)C(=O)O	-0.636067
36	O=C1OC(=O)C(F)=C1F	-0.067756

	Smiles	Reaction energy(eV)
37	O=C1OC(=O)C(F)=C1	-0.084355
38	O=C1OC(=O)C(N)=C1N	0.713211
39	O=C1OC(=O)C(=C1)N	0.681373
40	O=C1OC(=O)C(=C1)O	0.248712
41	O=C1OC(=O)C(O)=C1O	0.423137
42	O=C1OC(=O)C(S(=O)(=O)O)=C1S(=O)(=O)O	-0.477579
43	O=C1OC(=O)C(=C1)S(=O)(=O)O	-0.824263
44	C1=NCN=C1	-0.751035
45	O=C(O)C1=NCN=C1C(=O)O	-1.091596
46	C1=NCN=C1C(=O)O	-1.283562
47	FC1=NCN=C1F	-0.246263
48	C1=NCN=C1F	-0.507220
49	NC1=NCN=C1N	0.087893

2. Explanantory Analysis

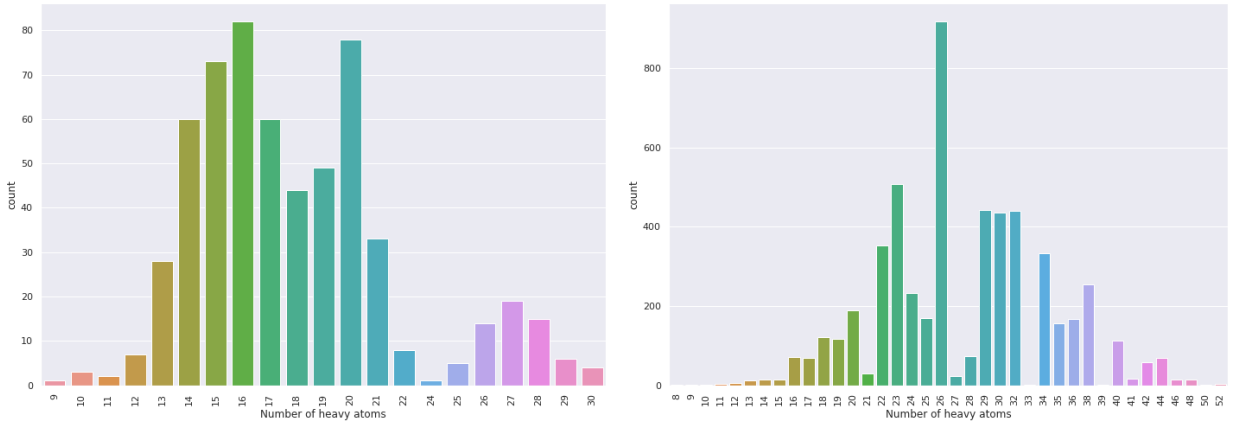


(a) Joint plot of reacton Energy and number of atoms in RedDB dataset

(b) Box plot of number of atoms in RedDB dataset

Figure S 1: Relationship of Number of atoms and Reaction energy in RedDB dataset a) Join plot of number of atoms and reaction energy and b) box plot of number of atoms

2.1. *Number of heavy atoms in RedDB*

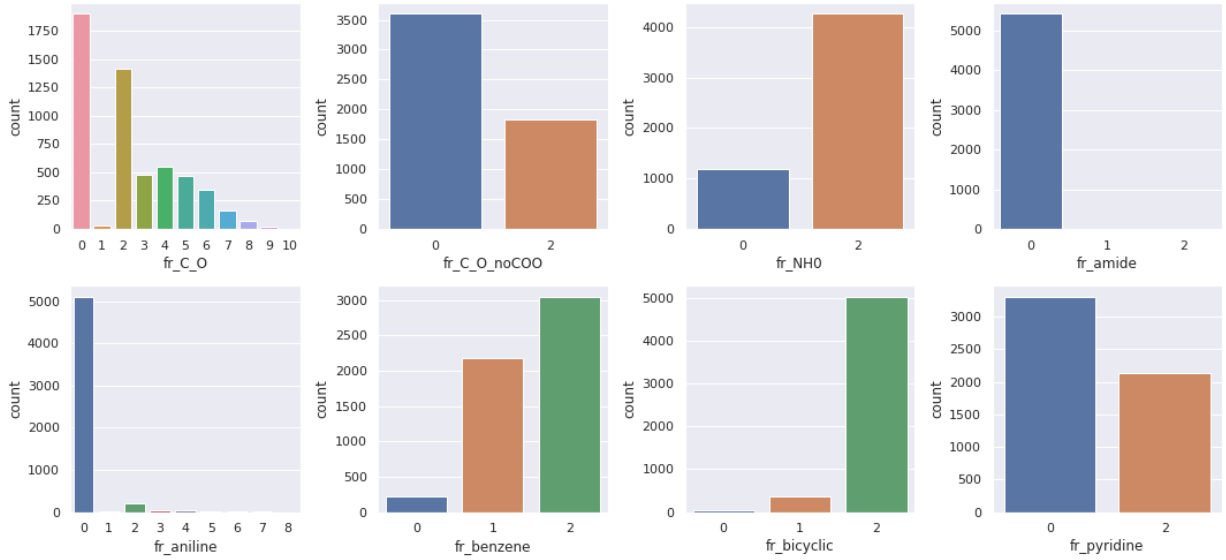


(a) number of heavy atoms catholyte molecules

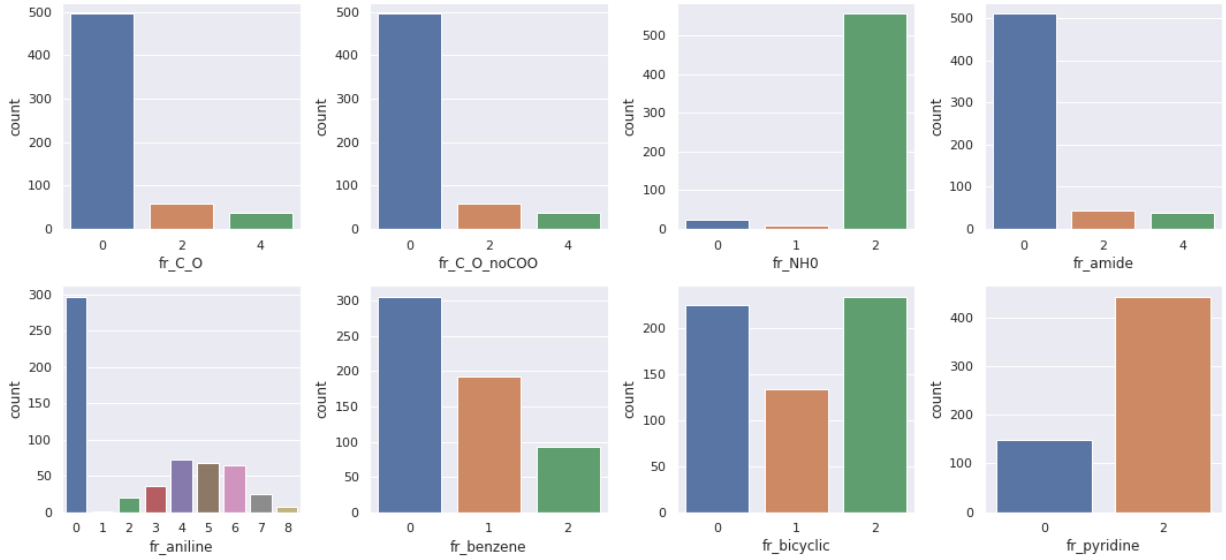
(b) Number of heavy atoms in anolyte molecules

Figure S 2: Number of heavy atoms in RedDB dataset a) number of heavy atoms catholyte molecules and b) Number of heavy atoms in anolyte molecules

2.2. *Types of functional groups in RedDB*



(a) Functional groups in the anolytes of RedDB dataset



(b) Functional groups in the catholytes of RedDB dataset

Figure S 3: The types and distributions of functional groups in RedDB dataset

2.3. *Pytorch_geometeric(Pyg) graph representation of RedDB from the SMILE Strings*

```
import os.path as osp
from pygdata import RedDB
from molfeatures import GenMolGraph, GenMolecules, GenMolFeatures
path = osp.join(osp.dirname(osp.realpath('__file__')),  'data', 'reddb')
```

```
mol_reddb = RedDB(root_dir=path,
                  name='reddb.csv',
                  smi_idx=-2,
                  target_idx=-1,pre_transform=GenMolFeatures()).shuffle()
```

2.4. *Target variable normalized to mean = 0 and std =1*

```
mol_reddb.data.y = mol_reddb.data.y*27.2114 # 1 Hartree=27.2114eV
r_mean = mol_reddb.data.y.mean()
r_std = mol_reddb.data.y.std()
mol_reddb.data.y = (mol_reddb.data.y - r_mean) / r_std
print("Normalized redox potential:\n",mol_reddb.data.y)
```

```
Normalized redox potential:
 tensor([[-0.4768],
        [ 0.1099],
        [-1.0686],
        ...,
        [ 0.7942],
        [ 0.3313],
        [ 0.0276]])
```

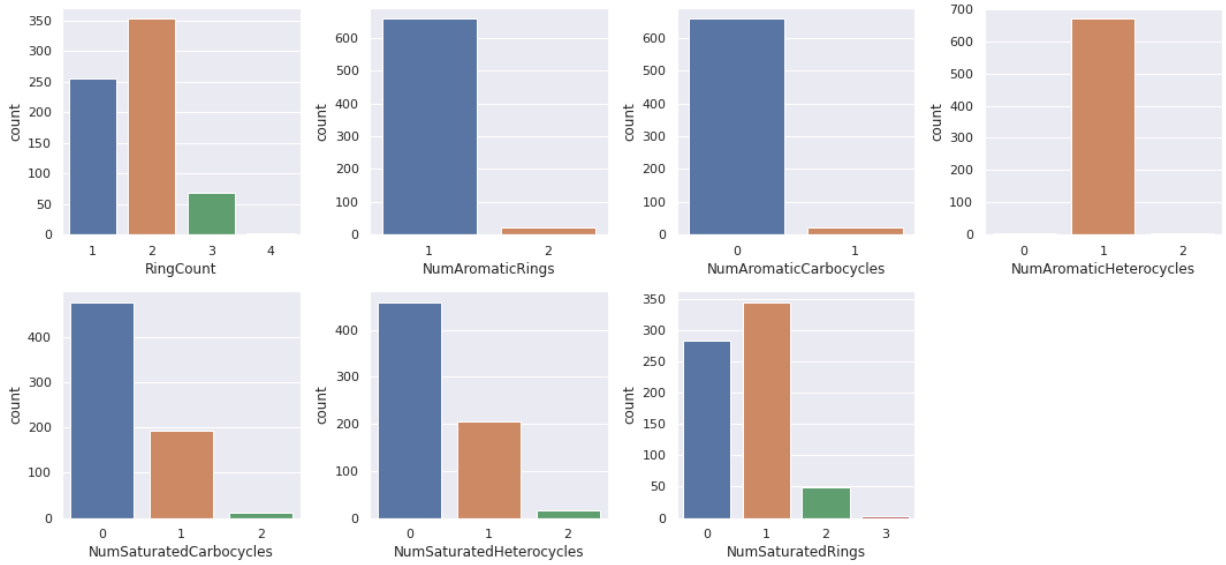
2.5. *Sample atomic attributes*

```
Sample atomic features in Pyg graph format:
 tensor([[0.0769, 0.0000, 0.0000,  ..., 0.0632, 0.2593, 0.2632],
        [0.0549, 0.0000, 1.0000,  ..., 0.0464, 0.3704, 0.2632],
        [0.0549, 0.0000, 1.0000,  ..., 0.0464, 0.3704, 0.2632],
        ...,
        [0.0000, 1.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
        [0.0000, 1.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
        [0.0000, 1.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000]])
```

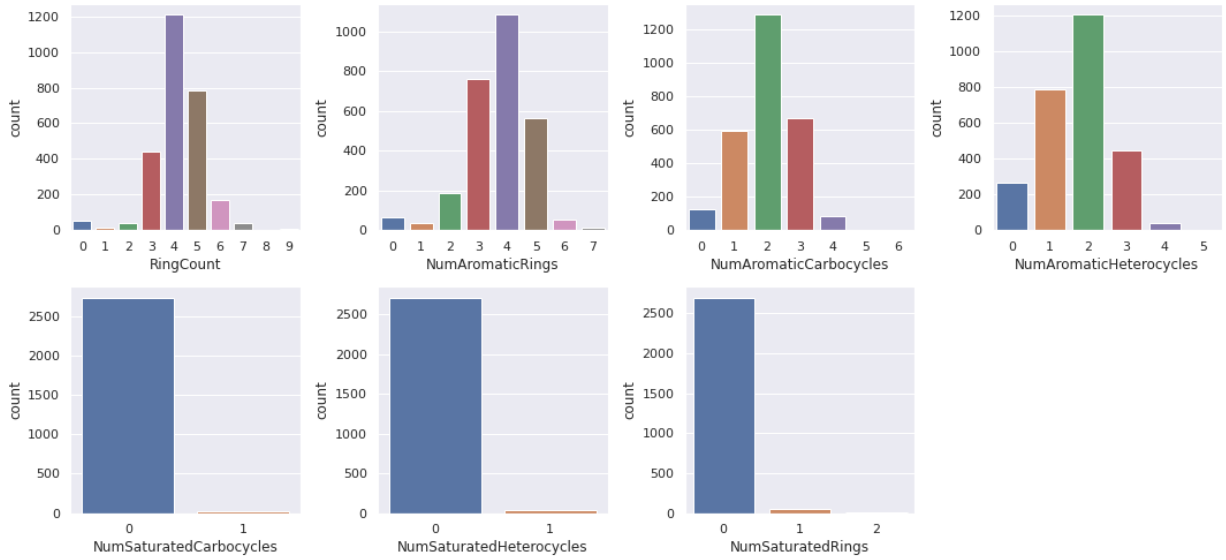
2.6. *Sample edge attributes*

```
Sample edge features in Pyg graph format:
 tensor([[0., 1., 0.,  ..., 0., 0., 0.],
        [0., 1., 0.,  ..., 0., 0., 0.],
        [1., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [1., 0., 0.,  ..., 0., 0., 0.],
        [1., 0., 0.,  ..., 0., 0., 0.],
        [1., 0., 0.,  ..., 0., 0., 0.]])
```

2.7. *Structural descriptors in RedDB dataset*



(a) Structural descriptors in catholyte screened dataset



(b) Structural descriptors in anolyte screened dataset

Figure S 4: Structure Structural descriptors of screened dataset

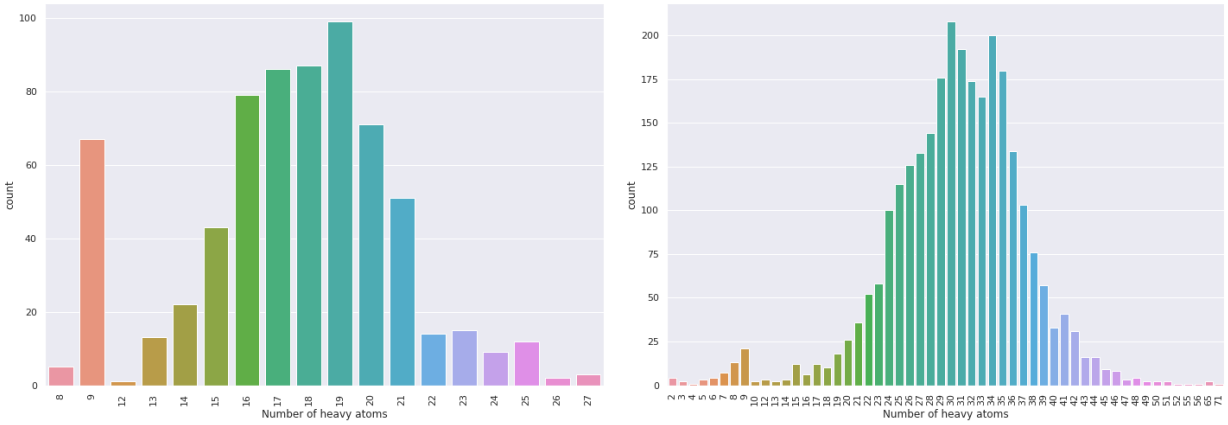
3. Screened Dataset

3.1. *Sample molecules from screened dataset*

	smiles	reaction_energy
0	O=C1C(c2ccccc2)=Nc2cc([N+](=O)[O-])c(O)cc21	-2.880693
1	CC1=C(C(=O)N(N1)C2=CC=CC=C2)C=C3C=NN=C3C4=CC(=...	-2.524888
2	C1=CC(=CC=C1C(=O)N=NC2=C3C=C(C=C(C3=NC2=O)Cl)C...	-2.375739

3	C1=CC(=O)C(=CN=C2C=CC3=NC(=NC3=C2)C4=C(C=C(C=C...	-2.350572
4	CONC1=C2C=C(C=CC2=NC1=O)[N+](=O)[O-]	-2.310877
5	CONC1=C2C=C(C=CC2=NC1=O)[N+](=O)[O-]	-2.310876
6	COC1=CC(=CN=C2C=CC3=NC(=NC3=C2)C4=CC=CC=C4)C(=...	-2.303406
7	C1=CC=C(C=C1)C2=NC3=CC(=NC=C4C=C(C=CC4=O)N(O)O...	-2.284782
8	C1=CC=C(C=C1)C2=NC(=C3C=CC(=O)C(=C3)N(O)O)N=C2...	-2.202916
9	C1=CC=C(C=C1)N2C(=O)C(=C(N2)C(=O)[O-])C=C3C=NC...	-2.197017
10	C1=CC=C(C=C1)N2C(=O)C(=C(N2)C(=O)[O-])C=C3C=NC...	-2.197017
11	C1=CC2=C(C(=C1)[N+](=O)[O-])C(=O)N=C2[O-]	-2.083223
12	C1=CC2=C(C(=C1)[N+](=O)[O-])C(=O)N=C2[O-]	-2.083223
13	COC1=CC=C(C=C1)N2C(=C(C(=O)NC2=O)C=C3C=NN=C3C4...	-2.070825
14	C1=CC2=NC(=C3C=C(C=CC3=O)N=CC4=C(C(=CC=C4)N(O)...	-2.065413
23452	CCCC[NH2+]CCOc1cccnc1C	2.294042
23453	COc1ccsc1CN[C@H]1CC[C@H]([NH+](C)C)C1	2.299948
23454	Cc1cnccc1CC[NH2+][C@@H](C(C)C)C1CC1	2.302506
23455	CC[NH+](CC)CC[C@H](N)c1cnccc1C	2.313252
23456	CC[NH2+]C[C@@H](Oc1cccnc1C)C(C)C	2.314913
23457	COc1ccsc1C[NH+](C)[C@H](C)C1(C)CC1	2.323198
23458	CC[C@H]1CCCC[C@@H]1Oc1ccc(C)nc1C[NH2+]C(C)C	2.336868
23459	Cc1ncccc1O[C@H](C)C[NH2+]CC(C)C	2.338219
23460	CCC(C)(C)[NH2+]CCOc1cccnc1C	2.368817
23461	CN[C@@H](c1cnccc1C)[C@]1([NH+](C)C)CCC[C@H](C)C1	2.793114
23462	CC(C)(C)[NH2+]Cc1ccc(OCC[NH+]2CCCCC2)cn1	2.794882
23463	CCC(CC)(CCO)CNc1cncc(Br)c1	2.813119
23464	CC[C@H](C)COc1ccc(C[C@H]([NH3+])CC)nc1	2.833529
23465	Cc1ccc(O[C@H](C)[C@@H](C)O)c(C[NH2+]CC(C)C)n1	2.854385
23466	CC[C@@H](C)CN(CC)c1nc2c(s1)[C@H]([NH2+]C)CC(C)...	2.873744

3.2. *Number of heavy atoms in screened datset*

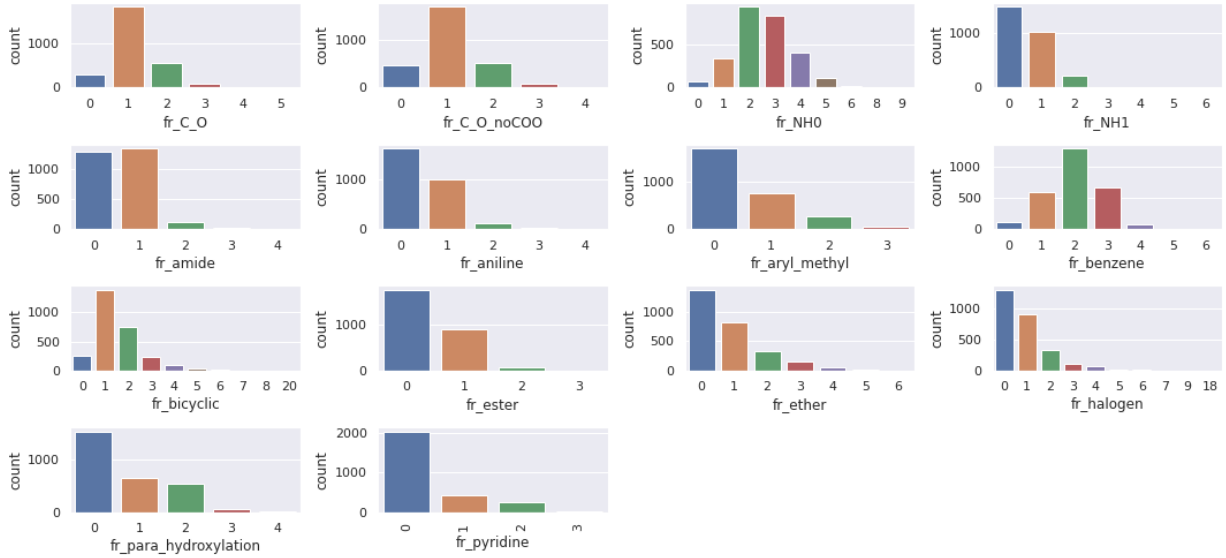


(a) number of heavy atoms catholyte molecules

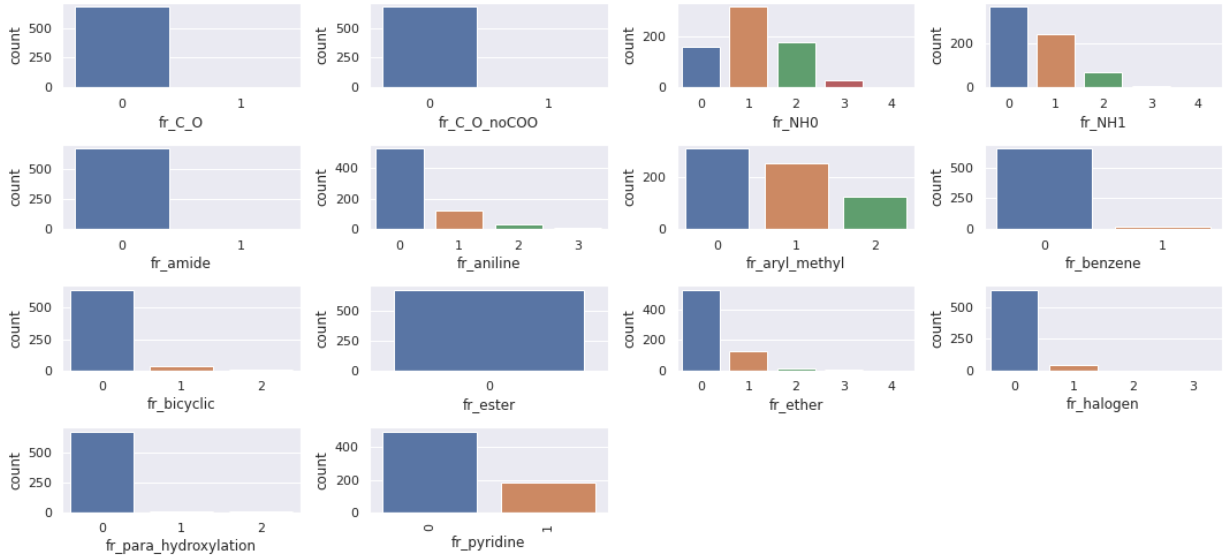
(b) Number of heavy atoms in anolyte molecules

Figure S 5: Number of heavy atoms in screened dataset a) number of heavy atoms catholyte molecules and b) Number of heavy atoms in anolyte molecules

3.3. *Types of functional groups in screened dataset*



(a) Functional groups in the anolytes of screened dataset



(b) Functional groups in the catholytes of screened dataset

Figure S 6: The types and distributions of functional groups in screen dataset

4. MolGAT Model Implementation

```
class MolGAT(torch.nn.Module):
    def __init__(self, node_features,
                 hidden_dim,
                 edge_features,
```

```
                num_heads,
                dropout,
                num_conv_layers,
                num_fc_layers):
        super(MolGAT, self).__init__()
        self.conv_list = torch.nn.ModuleList()
        self.bn_list = torch.nn.ModuleList()
        self.num_fc_layers = num_fc_layers

        self.bn_list.append(BatchNorm(hidden_dim))
        self.conv_list.append(MolGATConv(node_features, hidden_dim, edge_features, heads=num_heads)
        for i in range(num_conv_layers-1):
            self.conv_list.append(MolGATConv(hidden_dim, hidden_dim, edge_features, heads=num_heads

        self.fc_list = torch.nn.ModuleList()
        for i in range(num_fc_layers -1):
            if i == 0:
                self.fc_list.append(torch.nn.Linear(hidden_dim*2, hidden_dim*2))
            else:
                self.fc_list.append(torch.nn.Linear(hidden_dim*2, hidden_dim*2))

        self.fc_out = torch.nn.Linear(hidden_dim*2, 1)

        self.dropout = dropout

    def forward(self, x, edge_index, batch_index, edge_attr):
        for i, (conv, bn) in enumerate(zip(self.conv_list, self.bn_list)):
            x = F.relu(conv(x, edge_index, edge_attr))
            x = F.dropout(x, p=self.dropout, training=self.training)
            if i != (self.num_fc_layers-1):
                x = bn(x)

        x = torch.cat([gmp(x, batch_index),
                       gap(x, batch_index)], dim=1)

        for i, fc in enumerate(self.fc_list ):
            x = F.relu(fc(x))
            if i != (self.num_fc_layers-1):
                x = F.dropout(x, p=self.dropout, training=self.training)
```

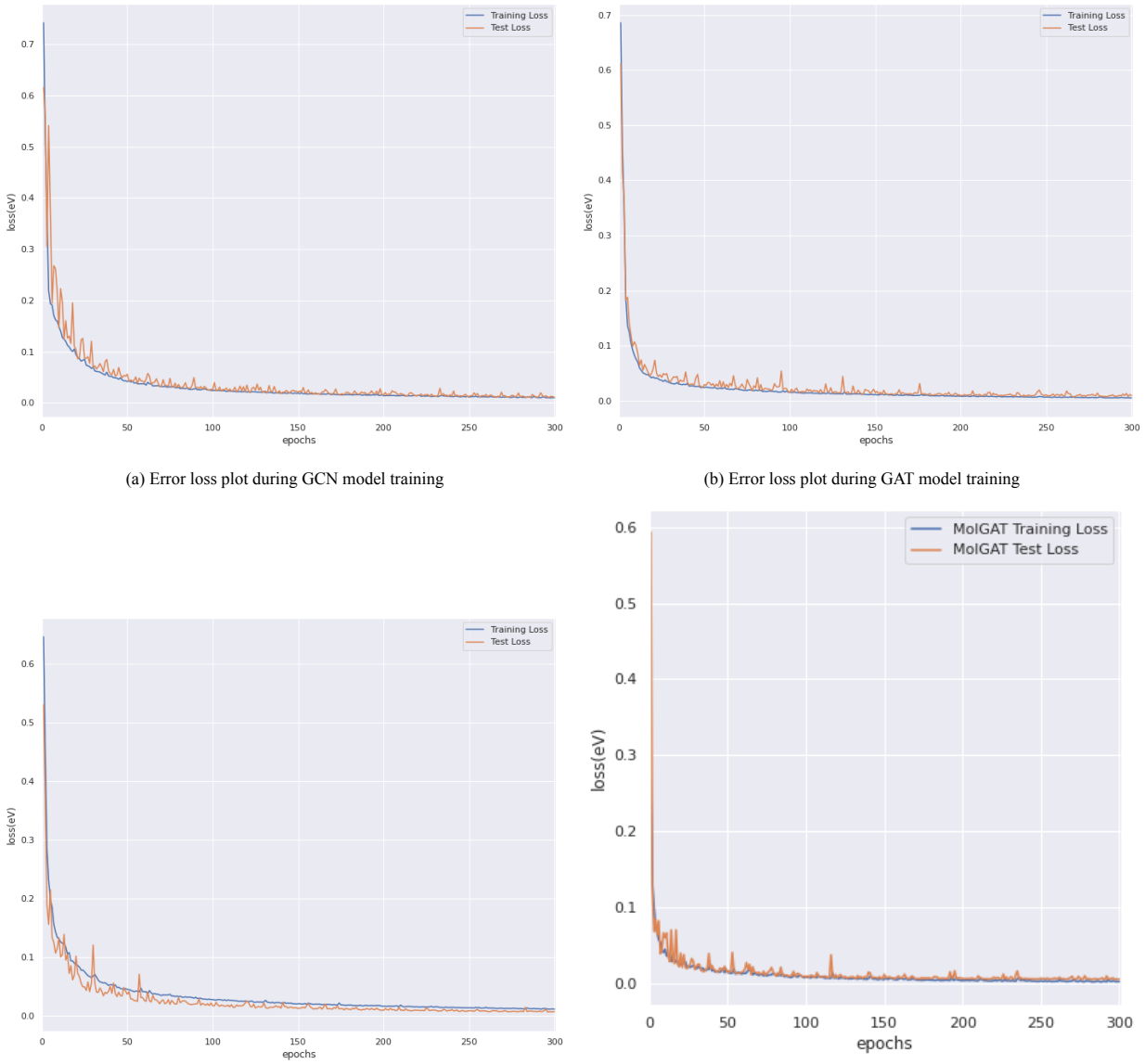
77

```
        x = self.fc_out(x)
        return x
# optimzed training parameters
class TrainArgs:
    edge_features = mol_reddb.data.edge_attr.shape[1]
    num_features=mol_reddb.num_features
    dropout=0.1
    num_fc_layers=3
    num_conv_layers=3
    num_heads=4
    hidden_dim=512
    batch_size = 192
args = TrainArgs()

# define the device and the molgat model

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = MolGAT(node_features=args.num_features,
                            hidden_dim=args.hidden_dim,
                            edge_features=args.edge_features,
                            num_heads=args.num_heads,
                            dropout=args.dropout,
                            num_fc_layers=args.num_fc_layers,
                            num_conv_layers=args.num_conv_layers).to(device)
## print model parameters
print(model)
print("Number of parameters: ", sum(p.numel() for p in model.parameters()))
```

```
MolGAT(
  (conv_list): ModuleList(
    (0): MolGATConv(99, 512, 12, heads=4)
    (1): MolGATConv(512, 512, 12, heads=4)
    (2): MolGATConv(512, 512, 12, heads=4)
  )
  (bn_list): ModuleList(
    (0): BatchNorm(512)
  )
  (fc_list): ModuleList(
    (0): Linear(in_features=1024, out_features=1024, bias=True)
    (1): Linear(in_features=1024, out_features=1024, bias=True)
  )
```

```
  (fc_out): Linear(in_features=1024, out_features=1, bias=True)
)
Number of parameters:  5219985
```

5. Training MolGAT Model



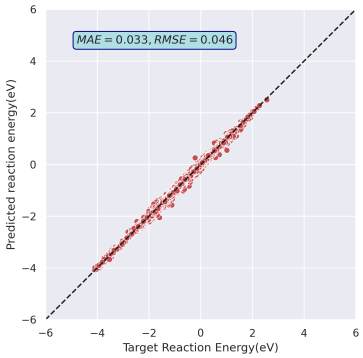
(a) Error loss plot during GCN model training

(b) Error loss plot during GAT model training

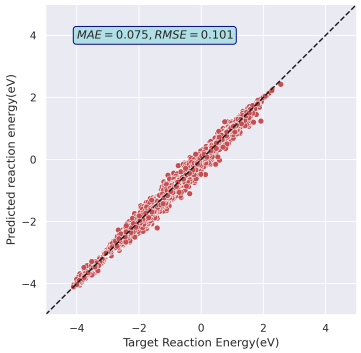
(c) Error loss plot during AttentiveFP model training

(d) Error loss plot for MolGAT model

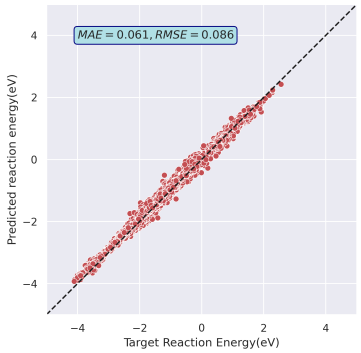
Figure S 7: Parity plots for Four different models a) Parity plot of MolGAT Model b) Parity plot of GCN Model c) Parity plot of GAT model d) Parity plot of AttentiveFP model



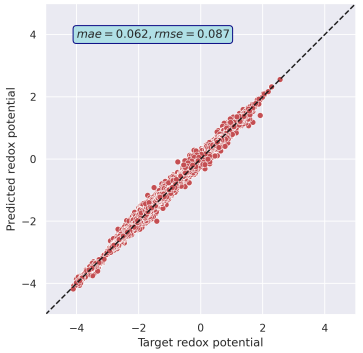
(a) Parity plot of MolGAT Model



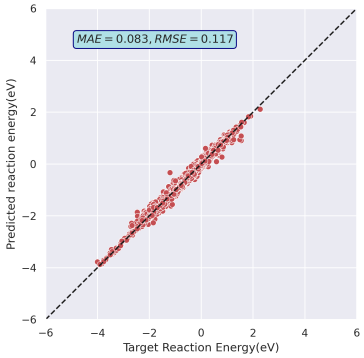
(b) Parity plot of GCN Model



(c) Parity plot of GAT model



(d) Parity plot of AttentiveFP model



(e) Parity plot of MPNN model

Figure S 8: Parity plots with their corresponding MAE and RMSE loss to benchmark four different models with a) Parity plot of MolGAT Model b) Parity plot of GCN Model c) Parity plot of GAT model d) Parity plot of AttentiveFP model e) Parity plot of MPNN Model