In [1]:

```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from tqdm import tqdm
import torch
import torch.nn as nn
import torchani
import matplotlib.pyplot as plt
```

In [2]:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#device = torch.device('cpu')
print(device)
```

```
cuda
```

In [3]:

```
def init_aev_computer():
    Rcr = 5.2
    Rca = 3.5
    EtaR = torch.tensor([16], dtype=torch.float, device=device)
    ShfR = torch.tensor([
        0.900000, 1.168750, 1.437500, 1.706250,
        1.975000, 2.243750, 2.512500, 2.781250,
        3.050000, 3.318750, 3.587500, 3.856250,
        4.125000, 4.393750, 4.662500, 4.931250
    ], dtype=torch.float, device=device)


    EtaA = torch.tensor([8], dtype=torch.float, device=device)
    Zeta = torch.tensor([32], dtype=torch.float, device=device)
    ShfA = torch.tensor([0.90, 1.55, 2.20, 2.85], dtype=torch.float, device=dev
    ShfZ = torch.tensor([
        0.19634954, 0.58904862, 0.9817477, 1.37444680,
        1.76714590, 2.15984490, 2.5525440, 2.94524300
    ], dtype=torch.float, device=device)

    num_species = 4
    aev_computer = torchani.AEVComputer(
        Rcr, Rca, EtaR, ShfR, EtaA, Zeta, ShfA, ShfZ, num_species
    )
    return aev_computer

aev_computer = init_aev_computer()
aev_dim = aev_computer.aev_length
print(aev_dim)
```

```
384
```

In [4]:

```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        return self.layers(x)
```

```
net_H = AtomicNet()
net_C = AtomicNet()
net_N = AtomicNet()
net_O = AtomicNet()
```

In [5]:

```
def load_ani_dataset(dspath):
    self_energies = torch.tensor([
        0.500607632585, -37.8302333826,
        -54.5680045287, -75.0362229210
    ], dtype=torch.float, device=device)
    energy_shifter = torchani.utils.EnergyShifter(None)
    species_order = ['H', 'C', 'N', 'O']

    dataset = torchani.data.load(dspath)
    dataset = dataset.subtract_self_energies(energy_shifter, species_order)
    dataset = dataset.species_to_indices(species_order)
    dataset = dataset.shuffle()
    return dataset

dataset = load_ani_dataset("./ani_gdb_s01_to_s04.h5")
# Use dataset.split method to do split
train_data, val_data, test_data = dataset.split(.8,.1,.1)
```

In [6]:

```
class ANITrainer:
    def __init__(self, model, batch_size, learning_rate, epoch, l2):
        self.model = model

        num_params = sum(item.numel() for item in model.parameters())
        print(f"{model.__class__.__name__} - Number of parameters: {num_params

        self.batch_size = batch_size
        self.optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate
        self.epoch = epoch

    def train(self, train_data, val_data, early_stop=True, draw_curve=True):
        ### Eric's comment: here you should pass in train_data, val_data, not


        self.model.train()

        # init data loader
        print("Initialize training data...")
        ### Eric's comment: call the collate().cache() here to init data loade
        train_data_loader = train_data_loader = train_data.collate(batch_size)

        # definition of loss function: MSE is a good choice!
        loss_func = torch.nn.MSELoss()

        # record epoch losses
        train_loss_list = []
        val_loss_list = []
        lowest_val_loss = np.inf

        for i in tqdm(range(self.epoch), leave=True):
            train_epoch_loss = 0.0
            for train_data_batch in train_data_loader:

                #computer energies
                species = train_data_batch['species'].to(device)
                coords = train_data_batch['coordinates'].to(device)
                true_energies = train_data_batch['energies'].to(device).float(
                _, pred_energies = model((species, coords))

                #compute loss
                batch_loss = loss_func(true_energies, pred_energies)


                # do a step
                ### Eric's comment: here you need to do optimization, follow t
                self.optimizer.zero_grad()
                batch_loss.backward()
                self.optimizer.step()

                batch_importance = len(train_data_batch) / len(train_data)


                ### Eric's comment: instead of directly using batch_loss, pleas
                ### batch_loss.detach().cpu().item(), please refer to the prev
                train_epoch_loss += batch_loss.detach().cpu().item() * batch_i

            # use the self.evaluate to get loss on the validation set
            val_epoch_loss = self.evaluate(val_data, draw_plot=False)

            # append the losses
            ### Eric's comment: train_epoch_loss should not divided by len(tra
            ### because it is already multiplied by the batch_importance
            train_loss_list.append(train_epoch_loss)
            val_loss_list.append(val_epoch_loss)

            if early_stop:
                if val_epoch_loss < lowest_val_loss:
                    lowest_val_loss = val_epoch_loss
                    weights = self.model.state_dict()

        if draw_curve:
            x_axis = np.arange(self.epoch)
            fig, ax = plt.subplots(1, 1, figsize=(5, 4), constrained_layout=Tru
            ax.set_yscale("log")
            # Plot train loss and validation loss
            ax.plot(x_axis, train_loss_list, label='Train')
            ax.plot(x_axis, val_loss_list, label='Validation')
            ax.legend()
            ax.set_xlabel("# Epoch")
            ax.set_ylabel("Loss")

        if early_stop:
            self.model.load_state_dict(weights)

        return train_loss_list, val_loss_list


    def evaluate(self, data, draw_plot=True):

        # init data loader
        ### Eric's comment: again, call the collate().cache() here to init data
        data_loader = data.collate(batch_size).cache()

        # init loss function
        loss_func = torch.nn.MSELoss()
        total_loss = 0.0

        if draw_plot:
            true_energies_all = []
            pred_energies_all = []

        with torch.no_grad():
            for batch_data in data_loader:
                ### Eric's comment: here the name train_data_batch is not appro
                ### necessarily not train data

                #compute energies
                species = batch_data['species'].to(device)
                coords = batch_data['coordinates'].to(device)
                true_energies = batch_data['energies'].to(device).float()
                _, pred_energies = model((species, coords))

                #computer loss
                batch_loss = loss_func(true_energies, pred_energies)

                ### Eric's comment: here should be len(data) because the argume
                ### is called data, not train_data
                batch_importance = len(batch_data) / len(data)
                ### Eric's comment: again, instead of directly using batch_loss
                ### batch_loss.detach().cpu().item(), please refer to the prev
                total_loss += batch_loss.detach().cpu().item() * batch_importan

                if draw_plot:
                    true_energies_all.append(true_energies.detach().cpu().numpy
                    pred_energies_all.append(pred_energies.detach().cpu().numpy

        if draw_plot:
            true_energies_all = np.concatenate(true_energies_all)
            pred_energies_all = np.concatenate(pred_energies_all)
            # Report the mean absolute error
            # The unit of energies in the dataset is hartree
            # please convert it to kcal/mol when reporting the mean absolute e
            # 1 hartree = 627.5094738898777 kcal/mol
            # MAE = mean(|true - pred|)
            hartree2kcalmol = 627.5094738898777
            mae = np.mean(np.abs((true_energies_all - pred_energies_all) * har
            fig, ax = plt.subplots(1, 1, figsize=(5, 4), constrained_layout=Tru
            ax.scatter(true_energies_all, pred_energies_all, label=f"MAE: {mae
            ax.set_xlabel("Ground Truth")
            ax.set_ylabel("Predicted")
            xmin, xmax = ax.get_xlim()
            ymin, ymax = ax.get_ylim()
            vmin, vmax = min(xmin, ymin), max(xmax, ymax)
            ax.set_xlim(vmin, vmax)
            ax.set_ylim(vmin, vmax)
            ax.plot([vmin, vmax], [vmin, vmax], color='red')
            ax.legend()

        return total_loss
```

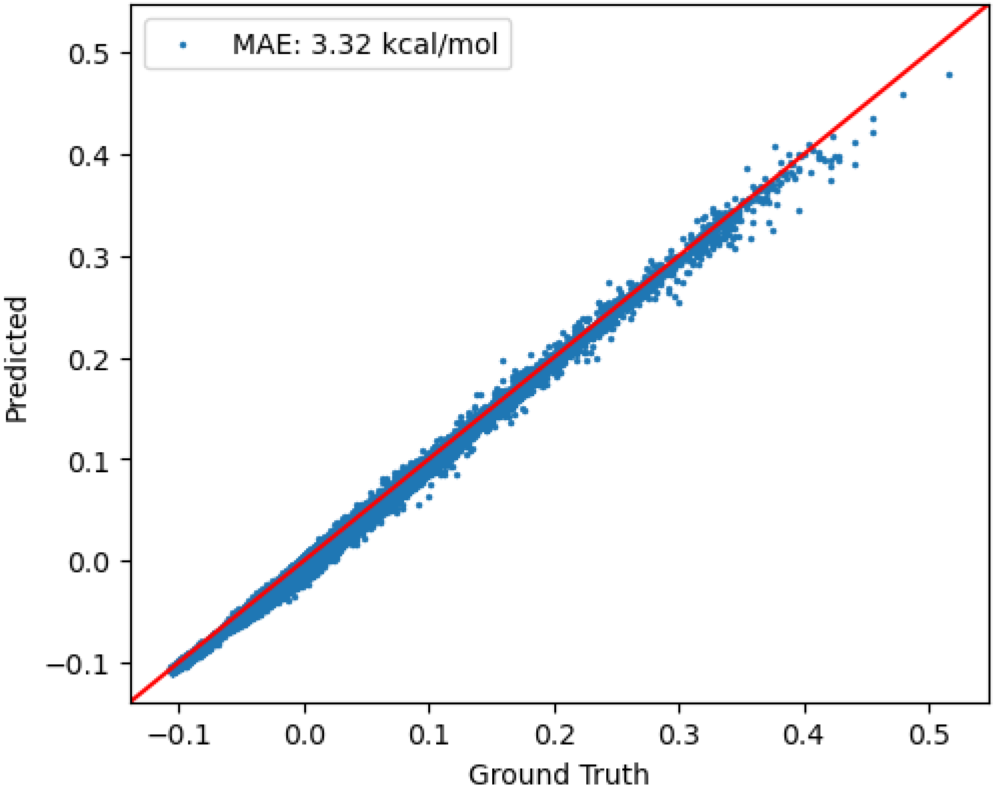
In [7]:

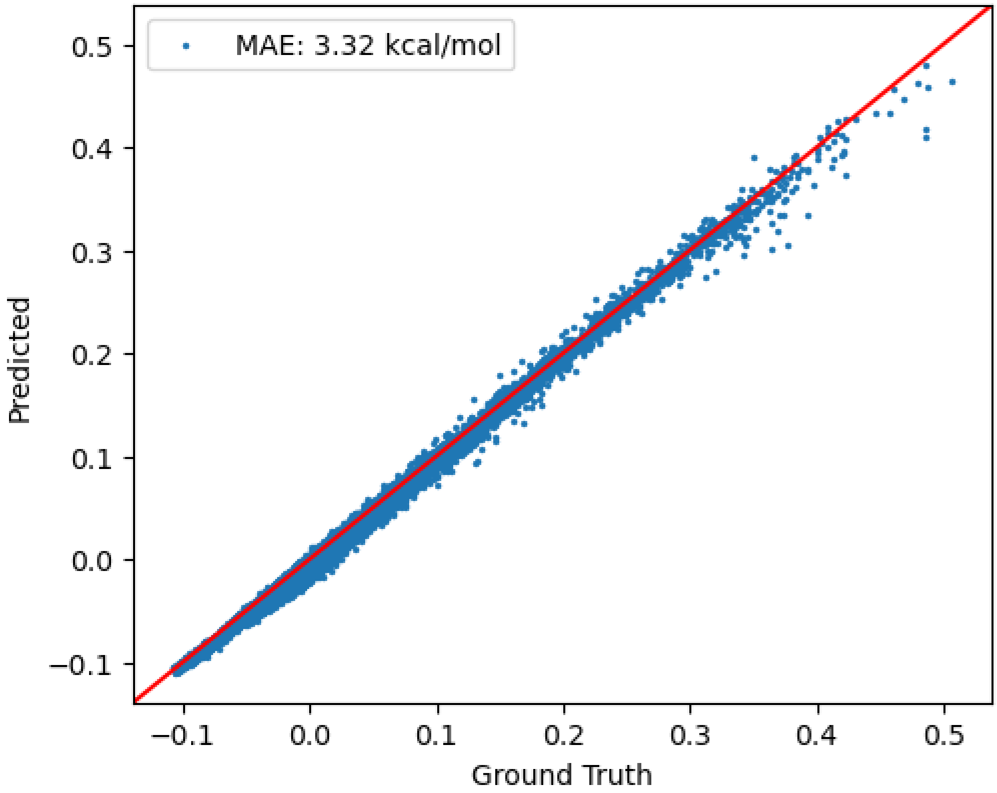
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
```

```
).to(device)

learning_rate = 1e-3
num_epochs = 30
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 30/30 [05:11<00:00, 10.37s/it]
```

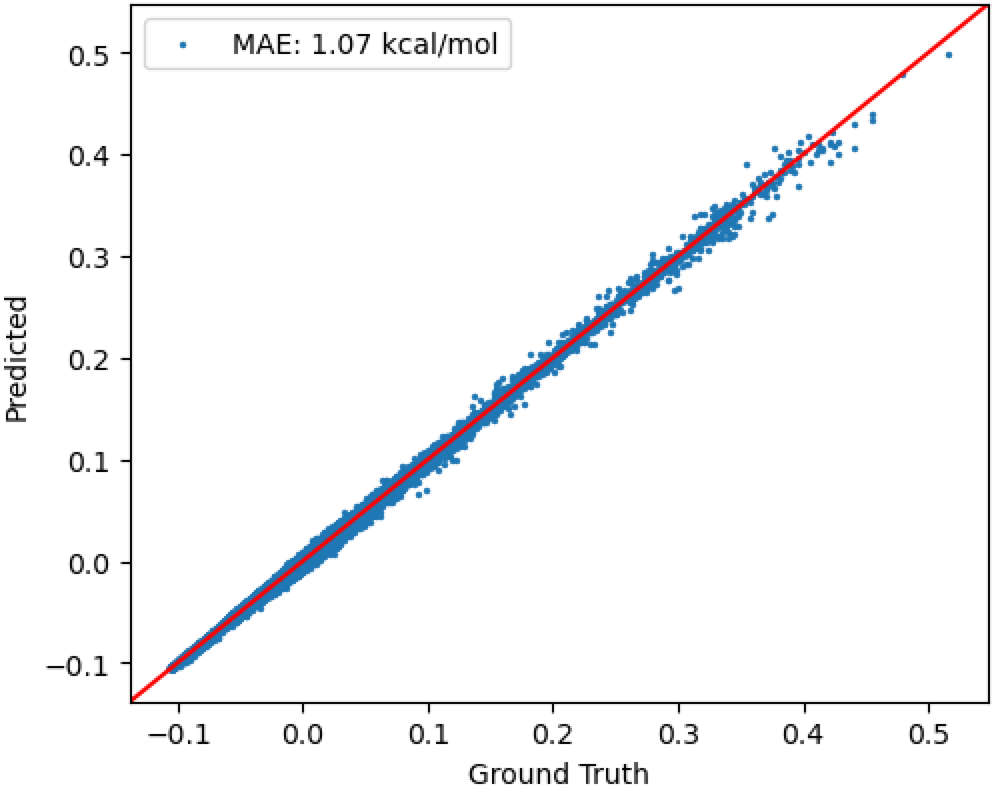


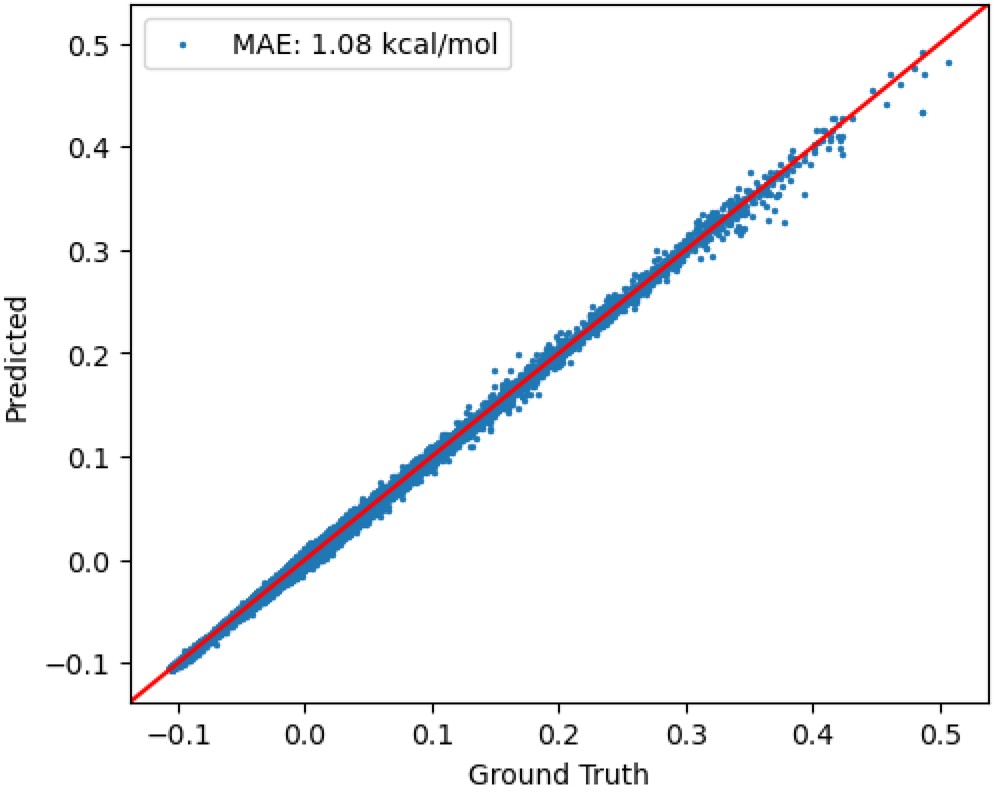


In [8]:

```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 30
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 30/30 [05:06<00:00, 10.23s/it]
```



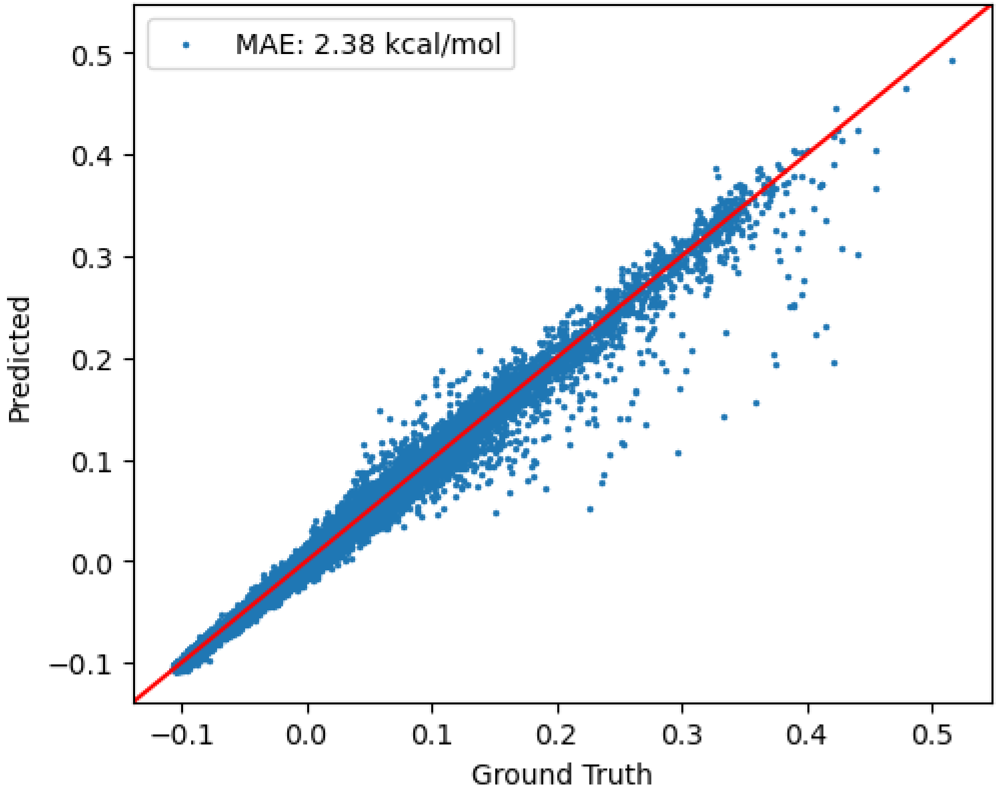


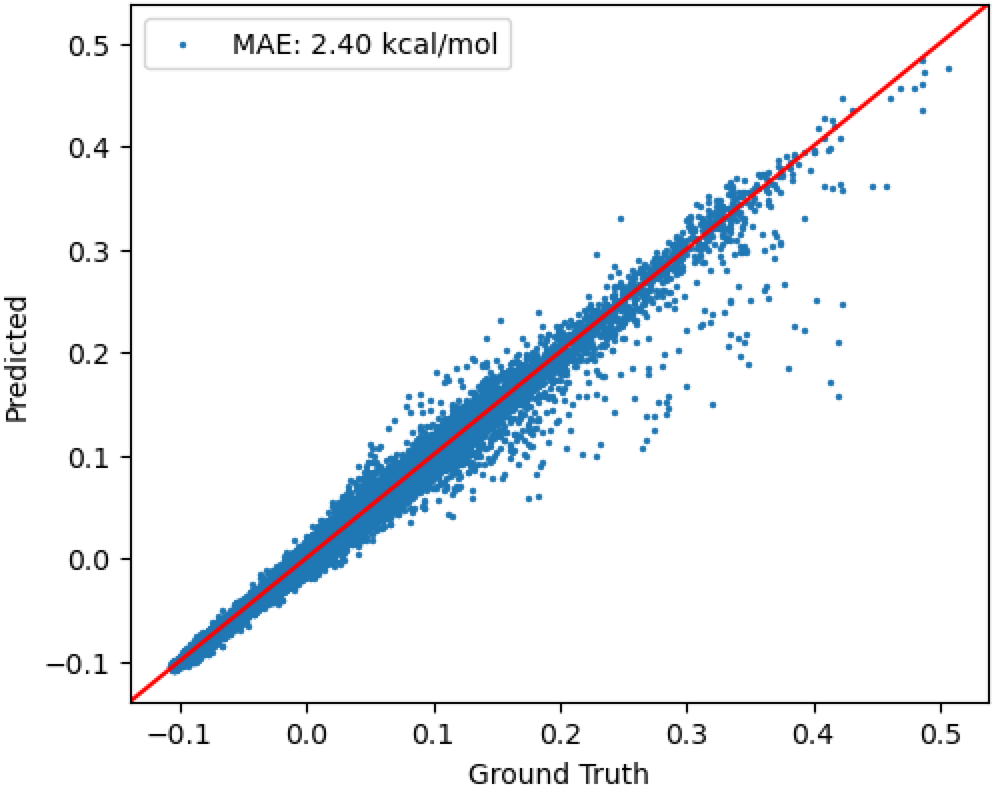
In [9]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-2
num_epochs = 30
```

```
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 30/30 [04:55<00:00,  9.87s/it]
```

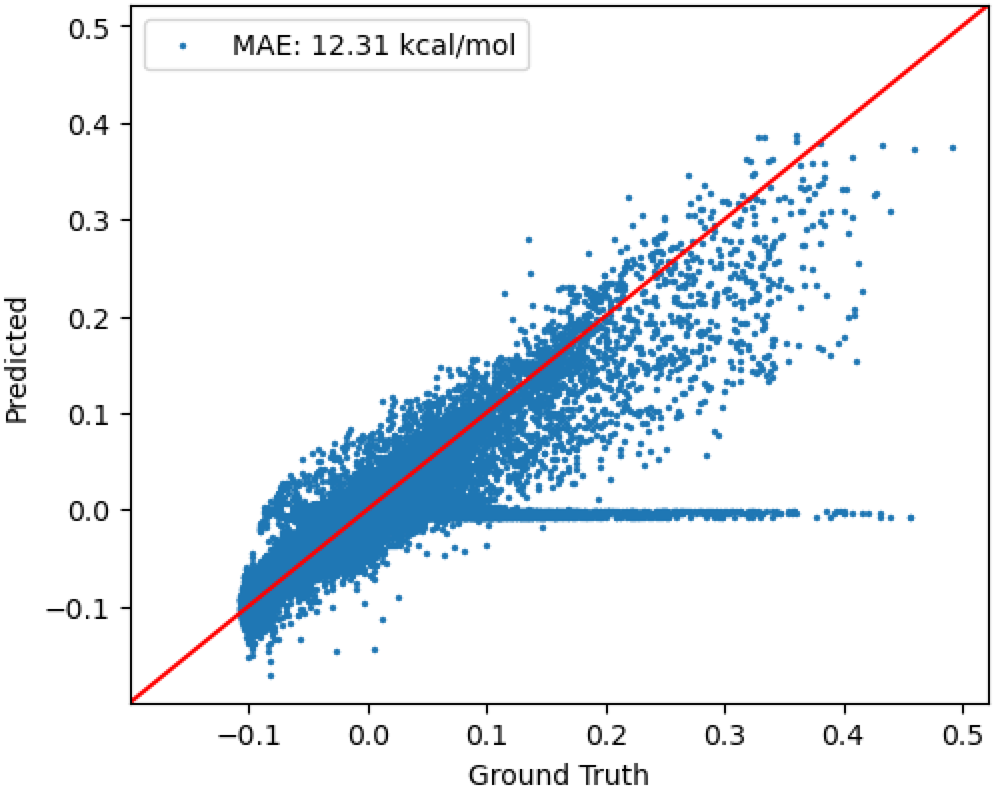


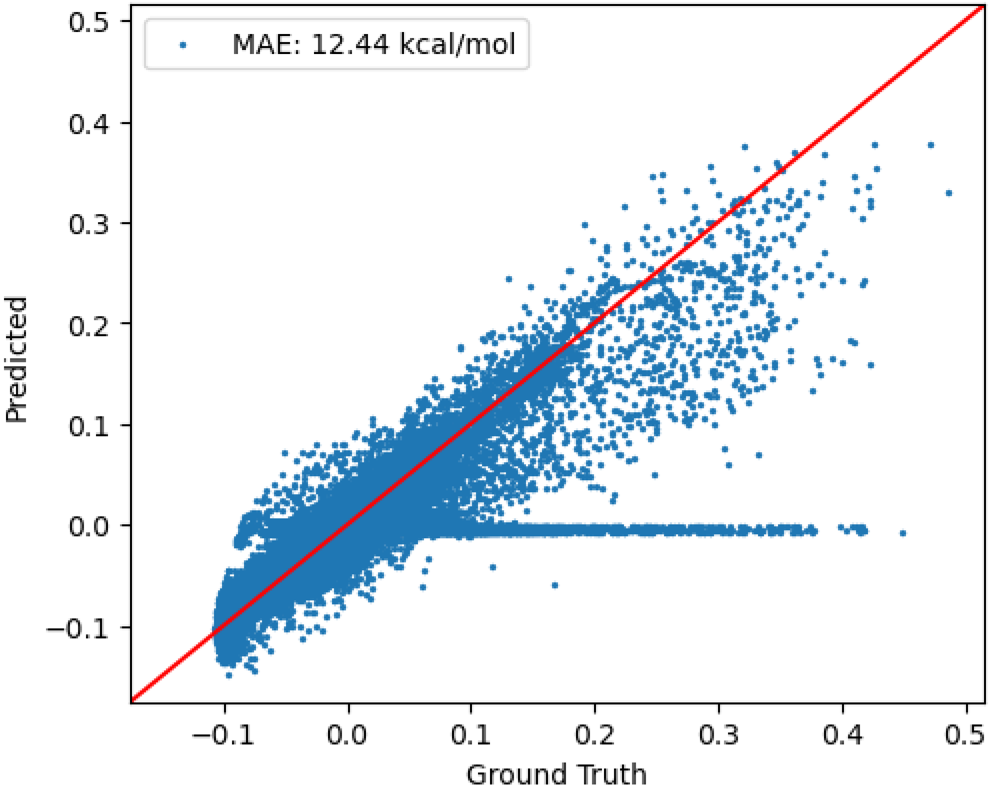


In [7]:

```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-1
num_epochs = 30
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 30/30 [04:59<00:00, 10.00s/it]
```

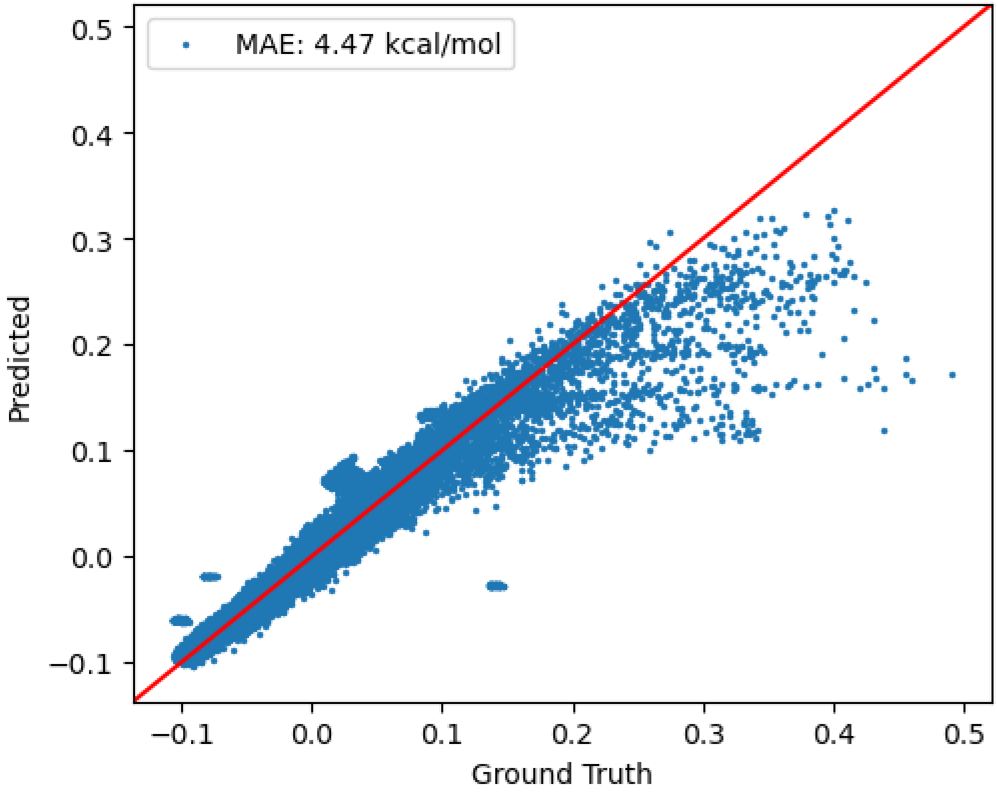


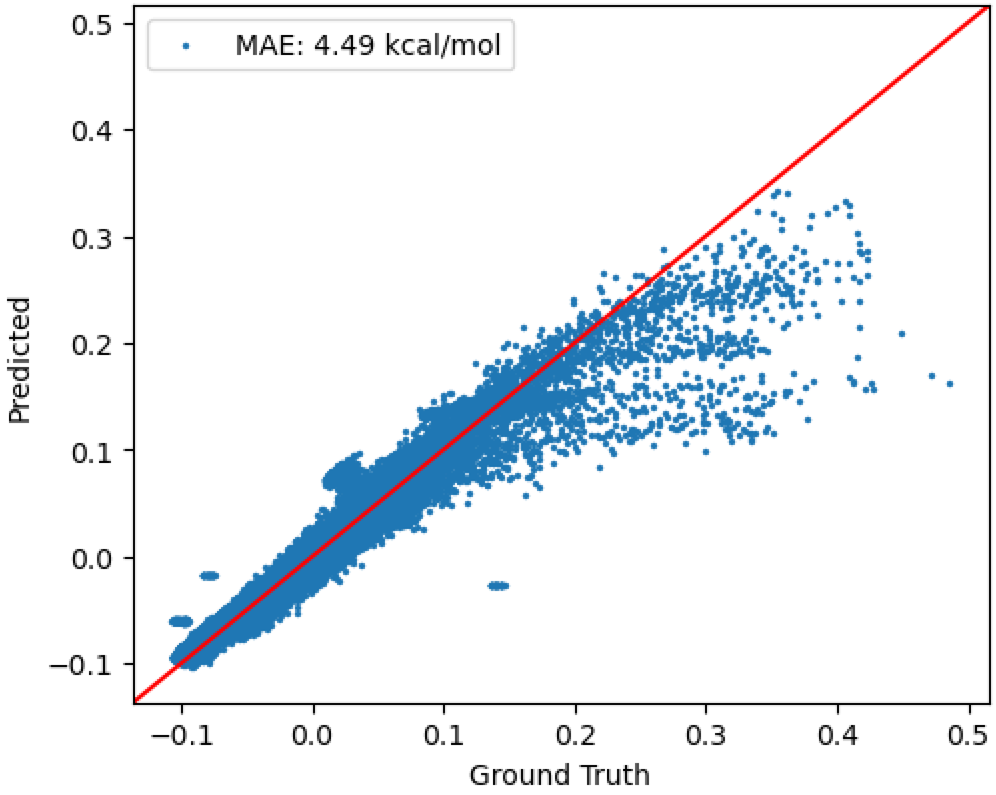


In [9]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-3
num_epochs = 30
```

```
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

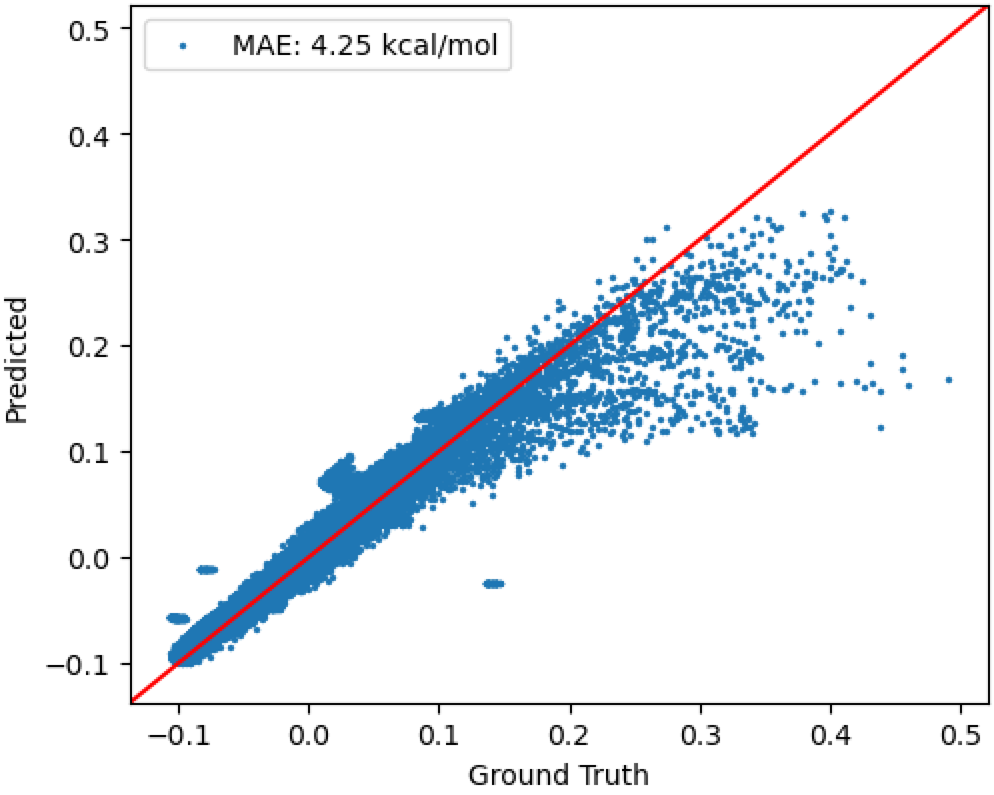
```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 30/30 [04:52<00:00,  9.76s/it]
```

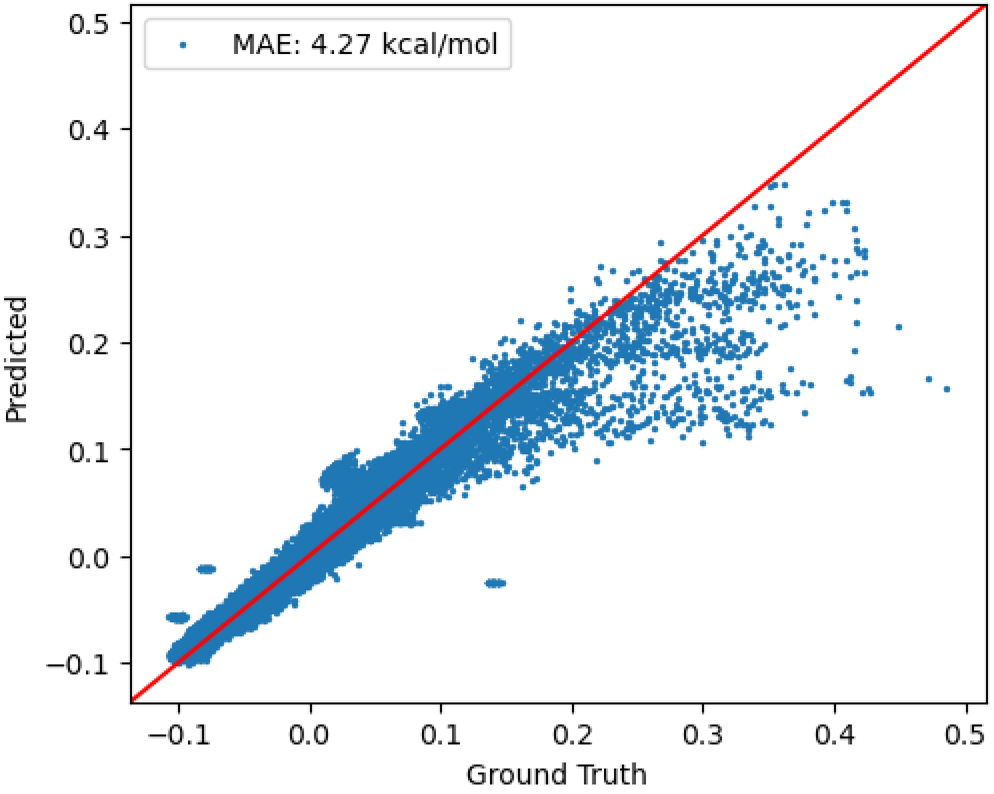




In [10]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 30
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 30/30 [04:52<00:00,  9.76s/it]
```



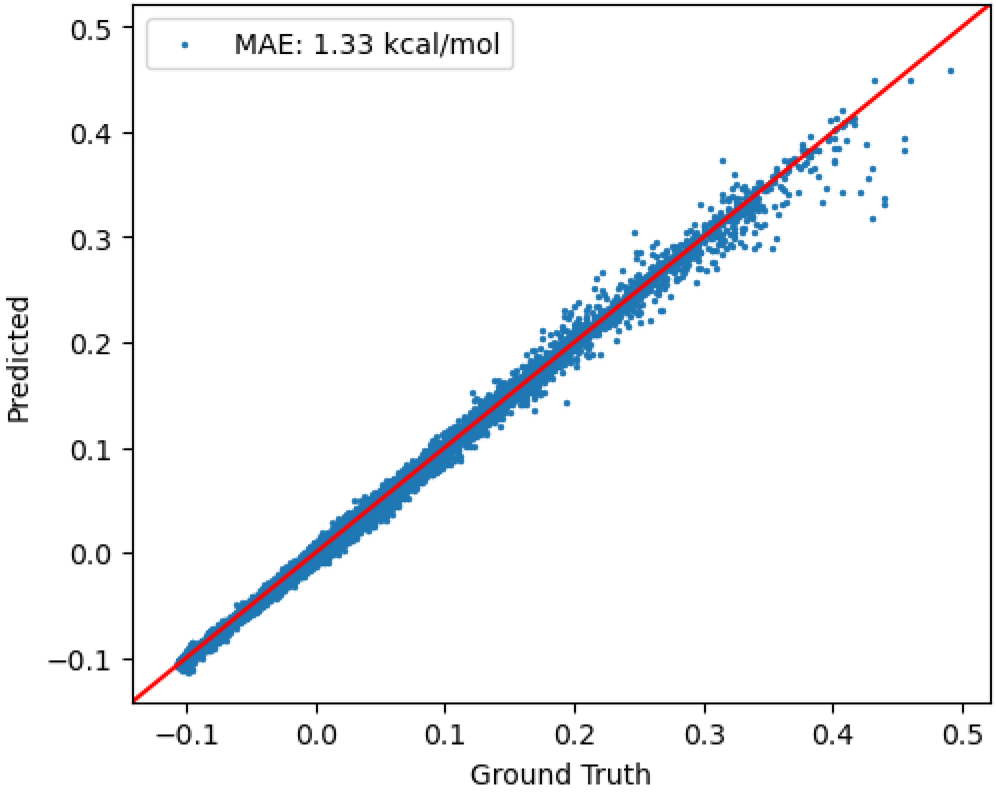


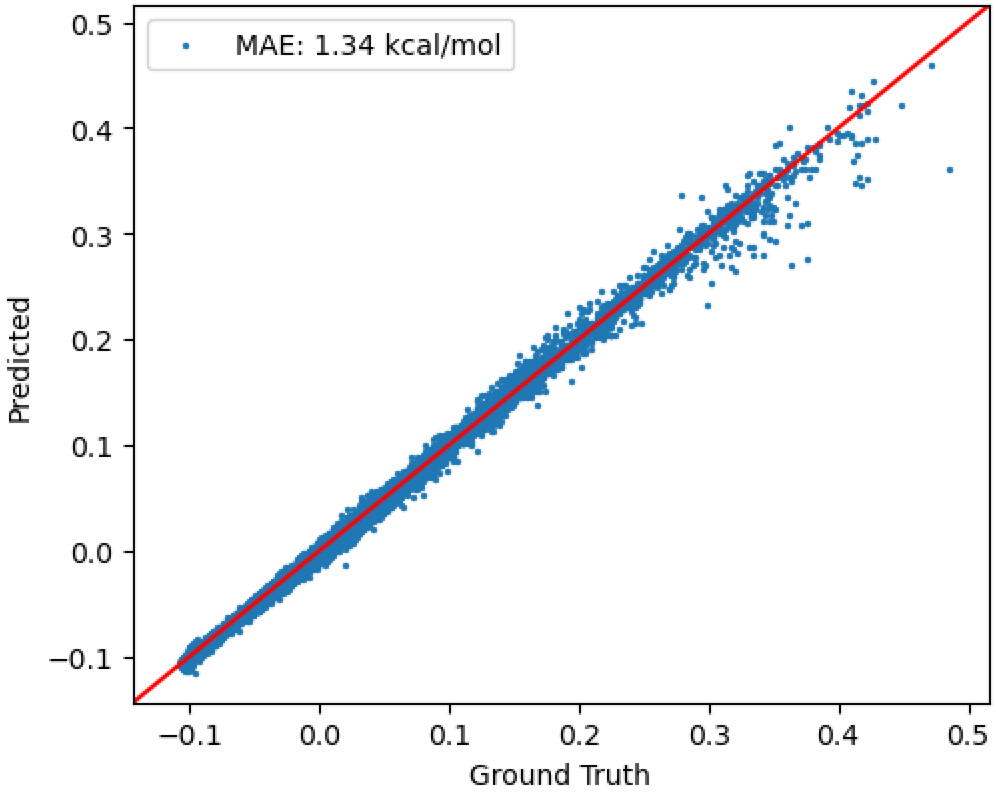
In [11]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 30
```

```
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 30/30 [04:57<00:00,  9.91s/it]
```

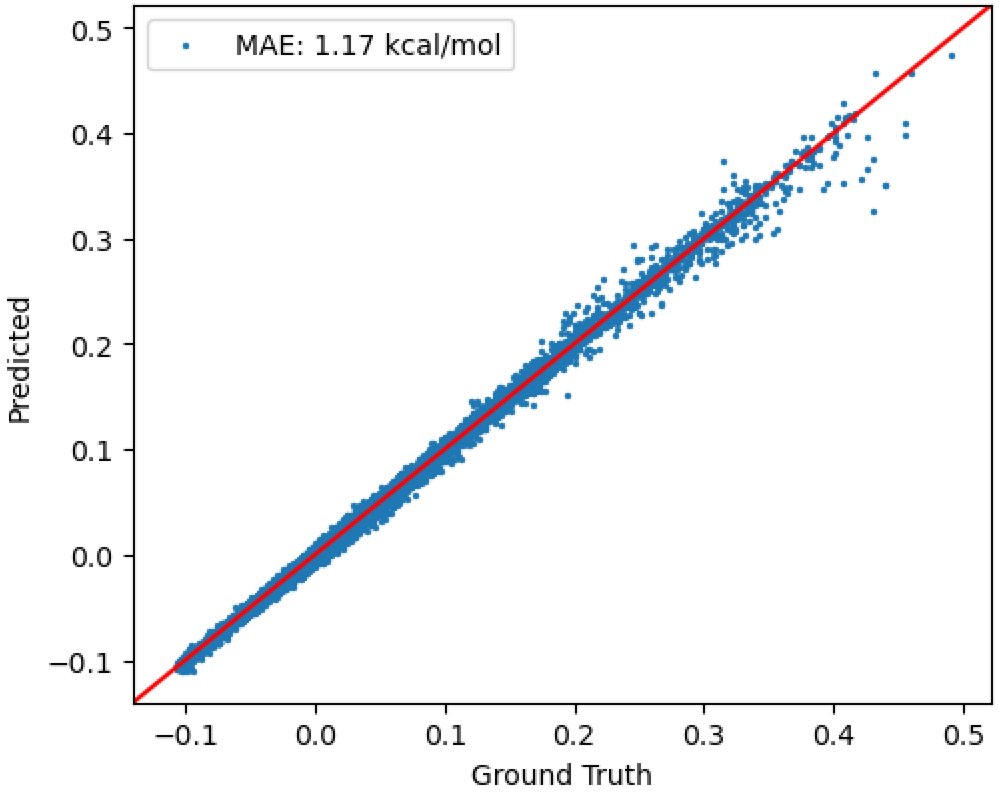


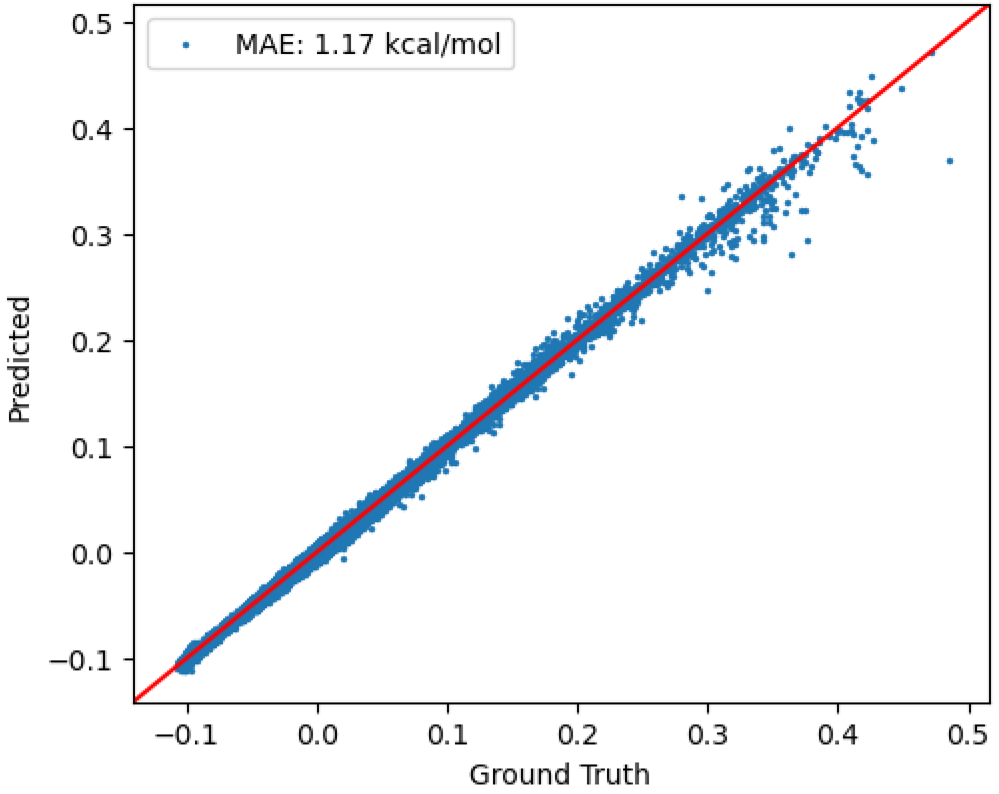


In [12]:

```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-5
num_epochs = 50
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 50/50 [07:57<00:00,  9.56s/it]
```





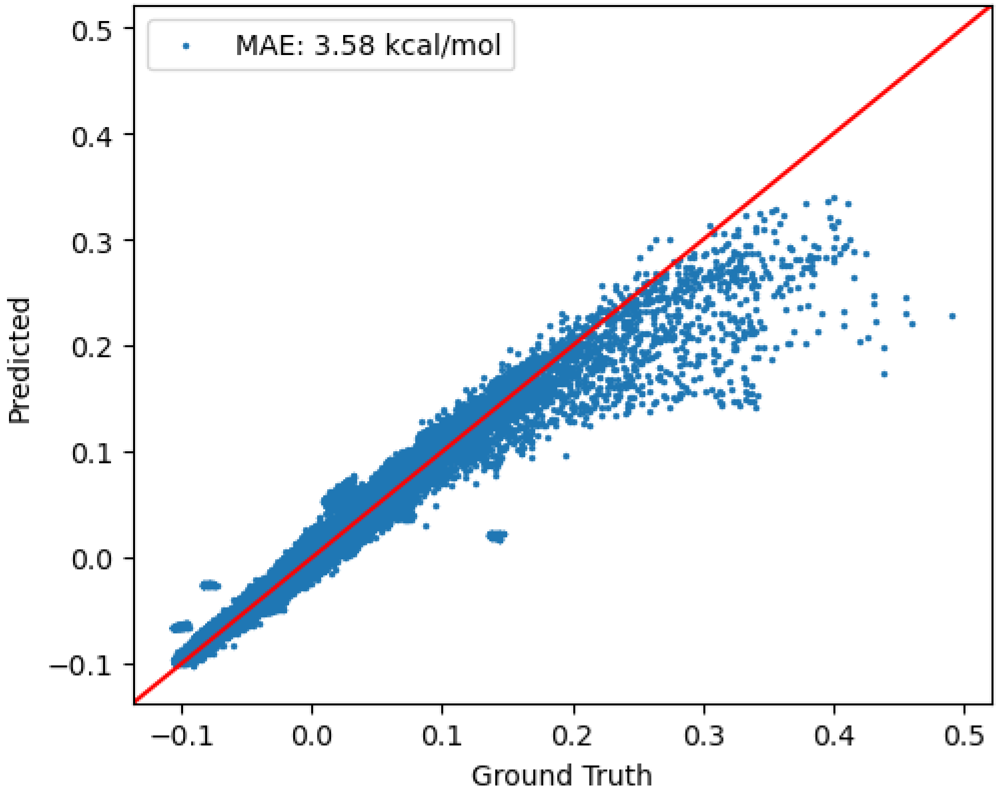
In [13]:

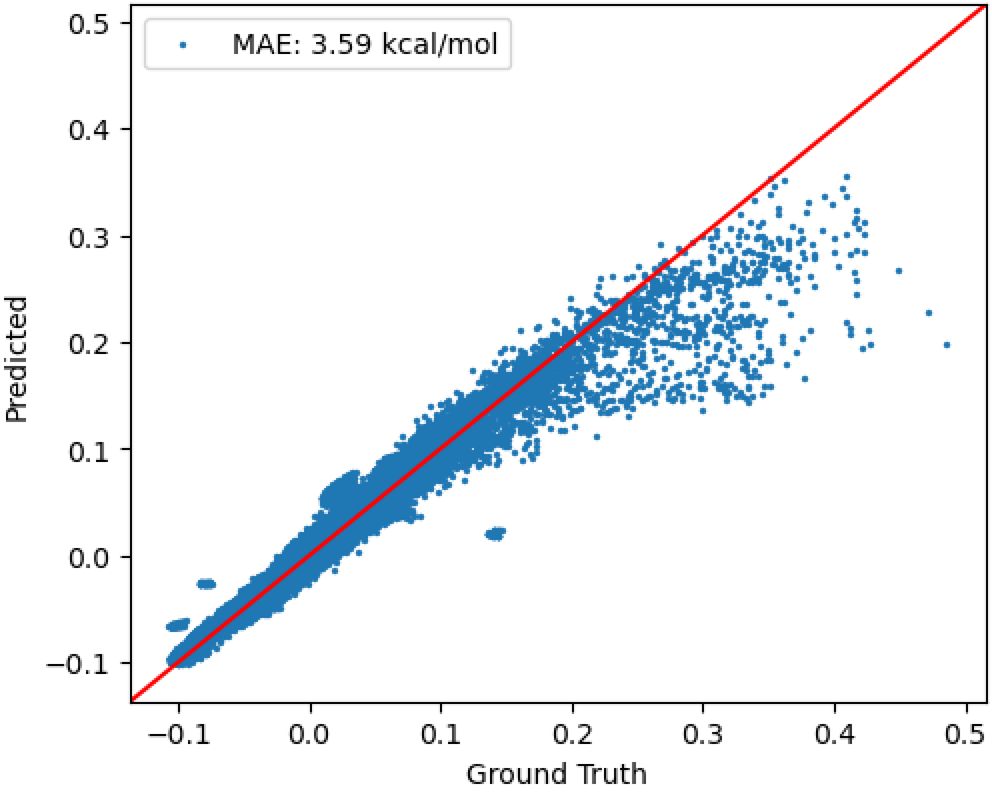
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-5
num_epochs = 50
```

```
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 50/50 [08:16<00:00,  9.94s/it]
```

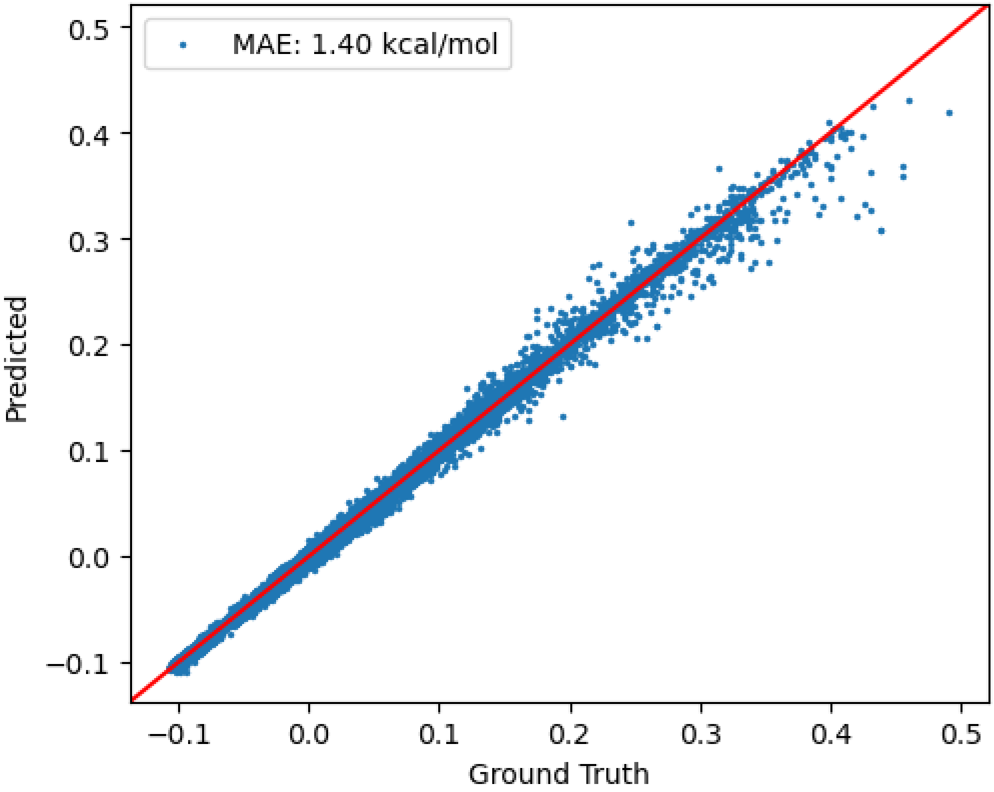


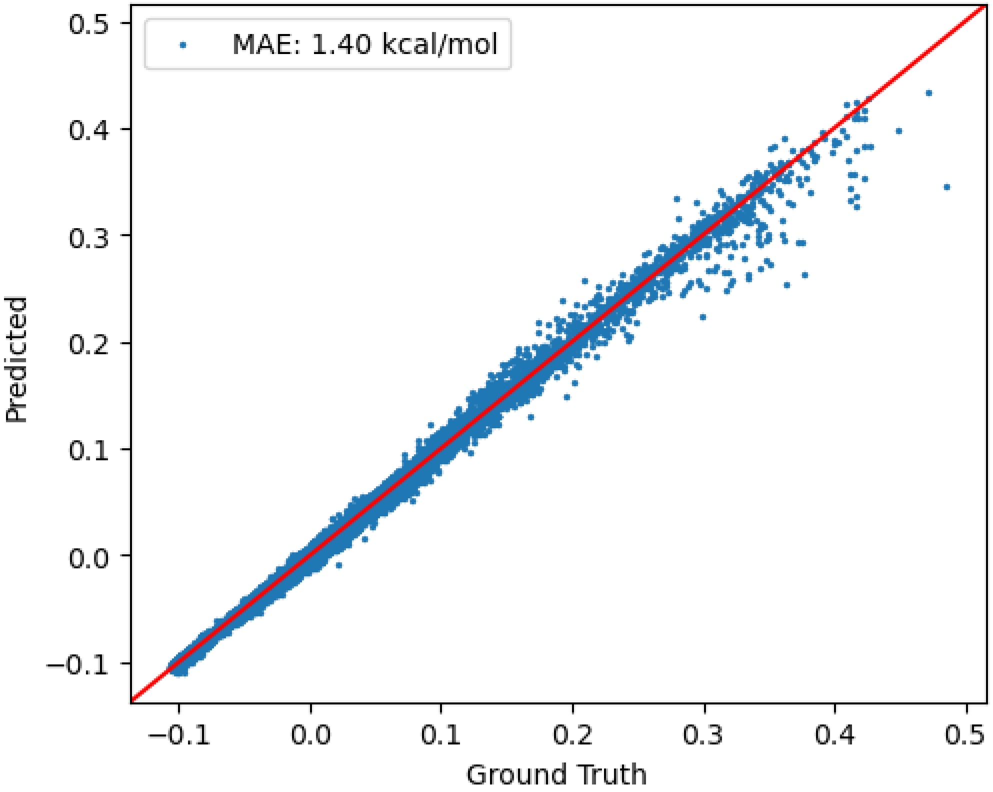


In [14]:

```python
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-5
num_epochs = 70
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 70/70 [11:27<00:00,  9.81s/it]
```





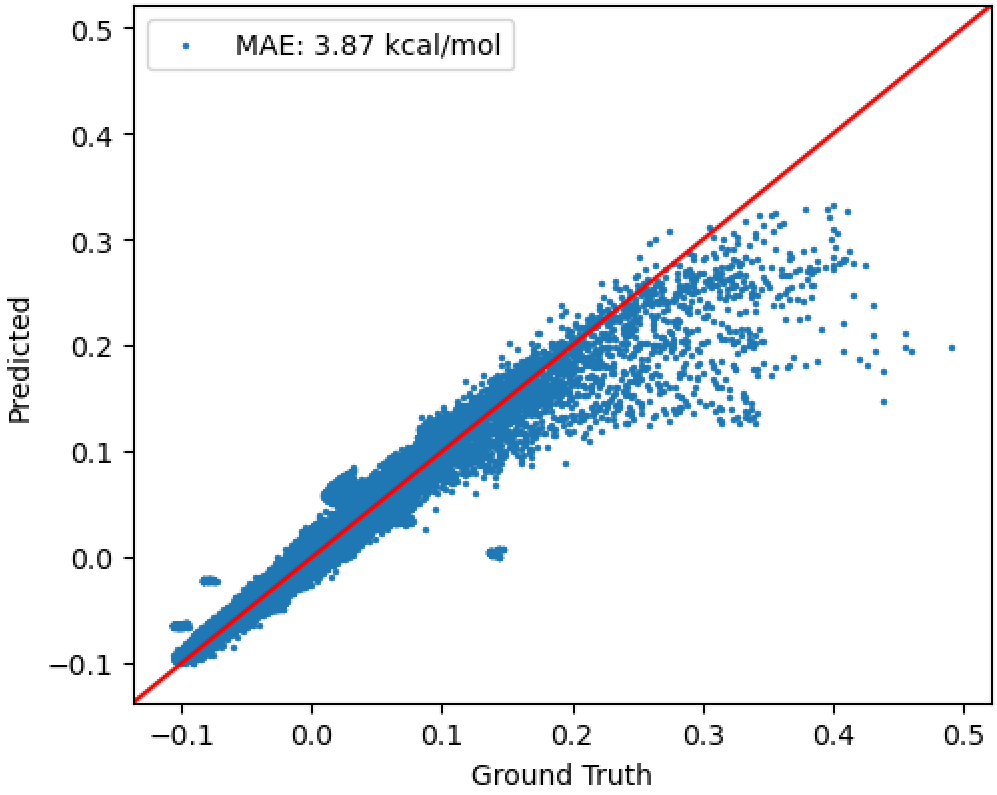
In [15]:

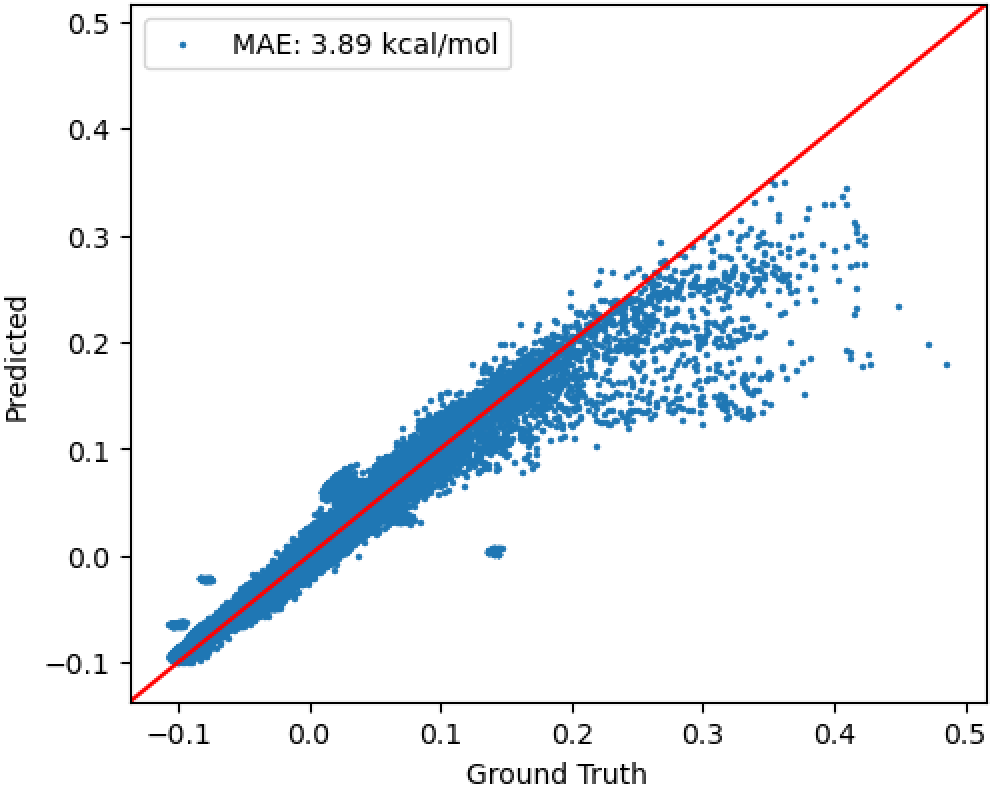
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-5
num_epochs = 50
```

```
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 50/50 [08:02<00:00,  9.64s/it]
```

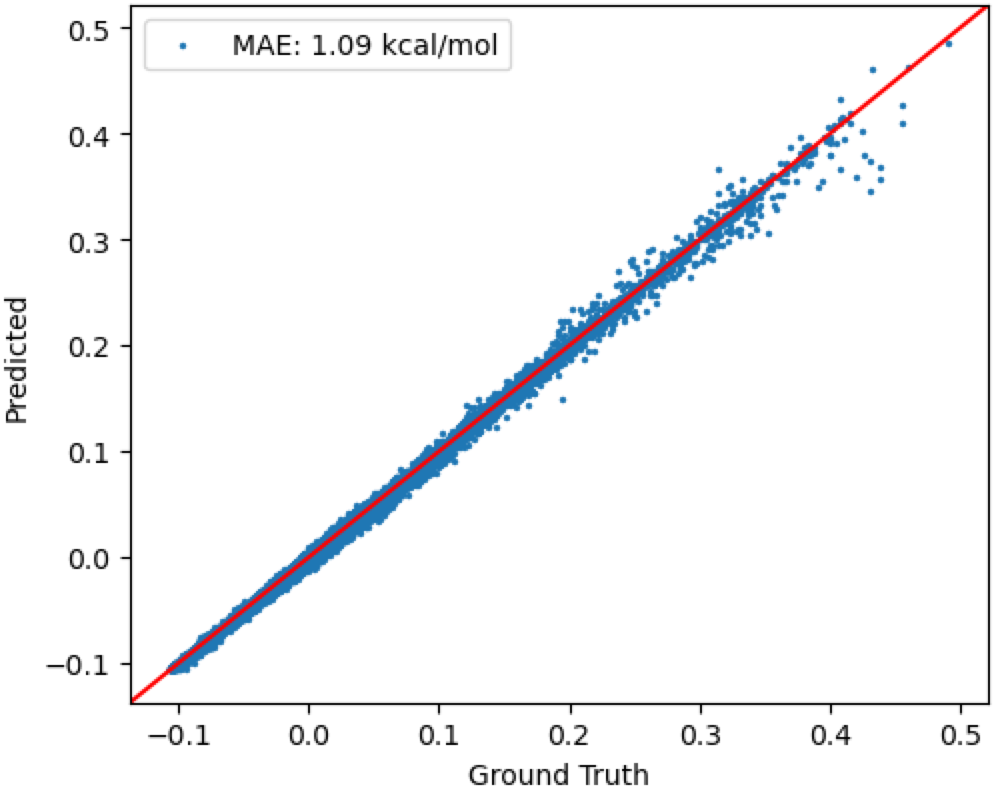


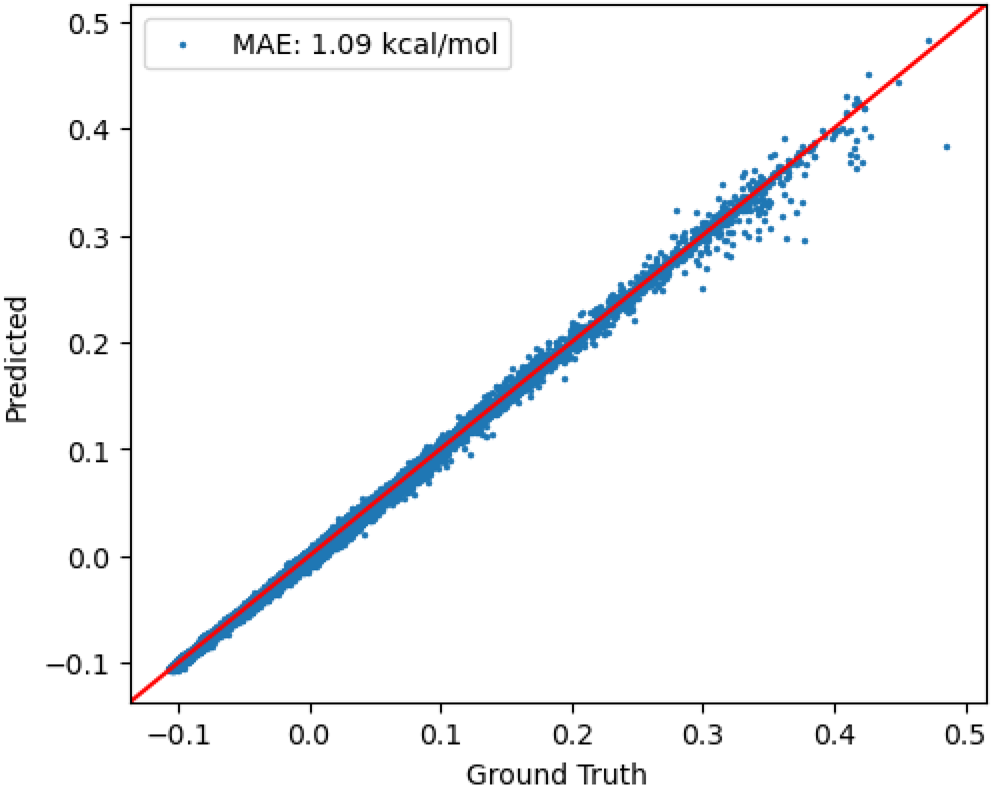


In [16]:

```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 50
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 50/50 [08:04<00:00,  9.69s/it]
```





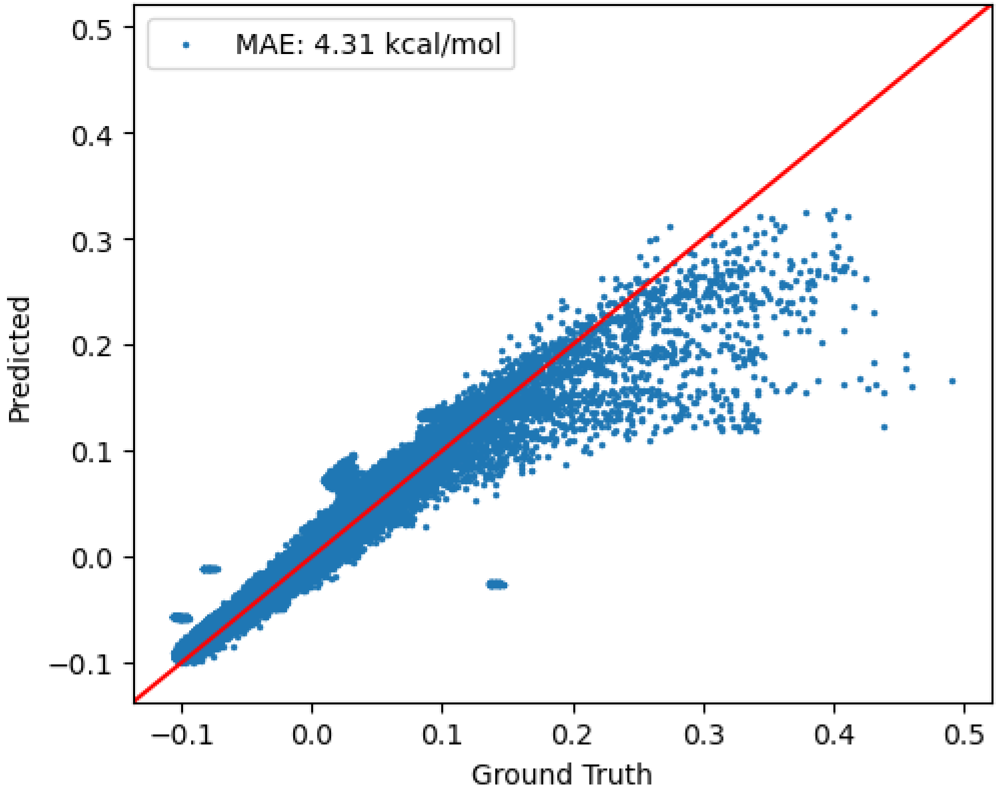
In [17]:

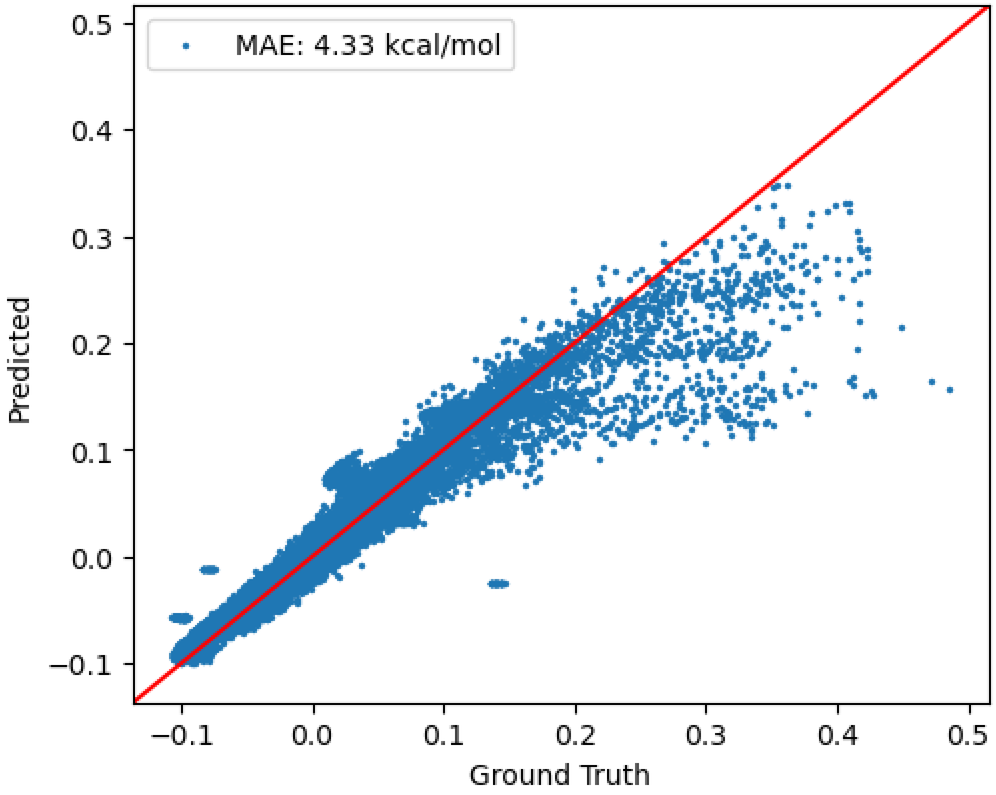
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 50
```

```
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
```

```
100%|██████████| 50/50 [08:18<00:00,  9.98s/it]
```



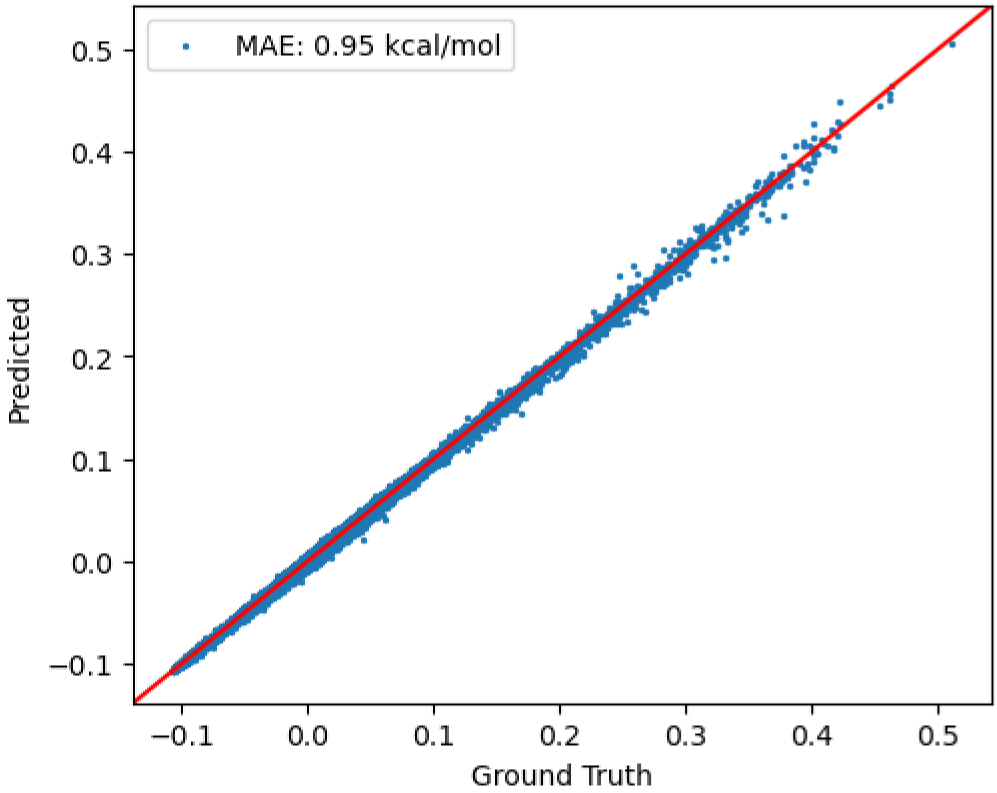


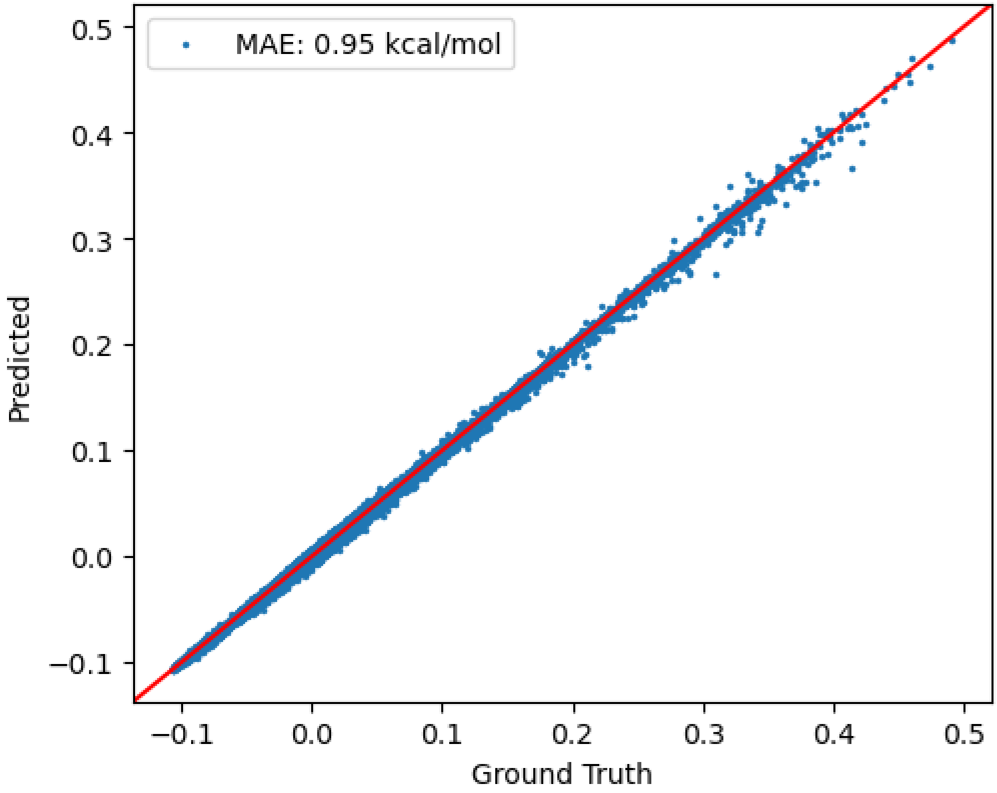
In [10]:

```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        return self.layers(x)

net_H = AtomicNet()
net_C = AtomicNet()
net_N = AtomicNet()
net_O = AtomicNet()

ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-3
num_epochs = 30
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=Fals
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 526340
Initialize training data...
```

```
100%|██████████| 30/30 [05:59<00:00, 11.99s/it]
```

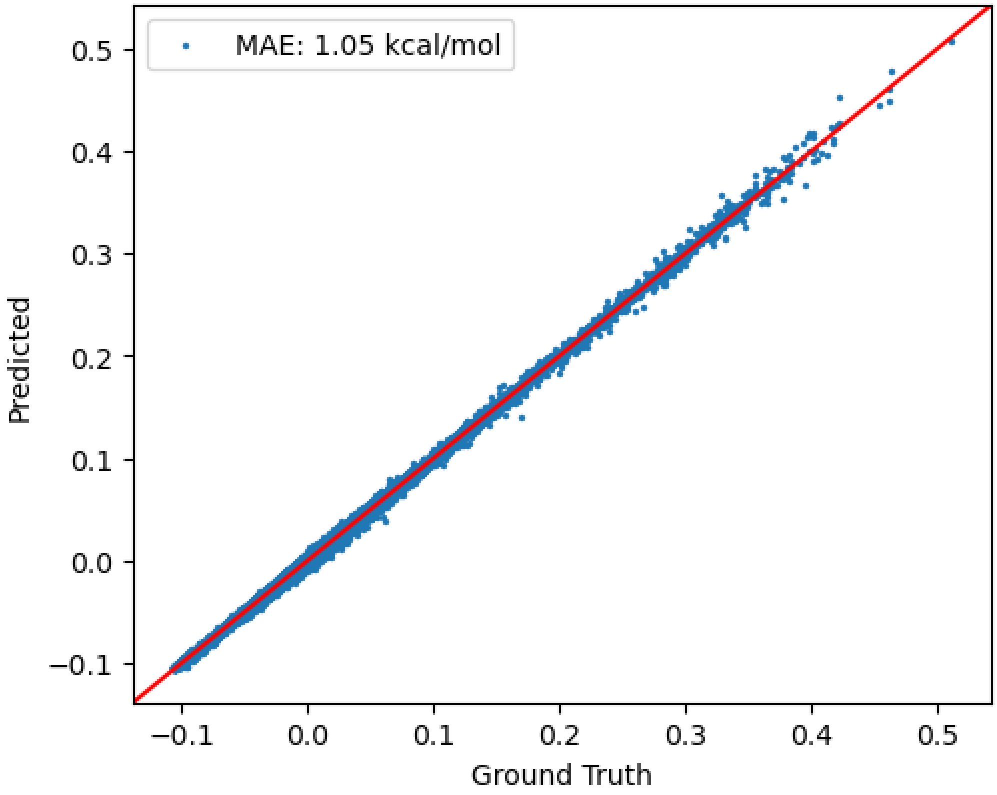


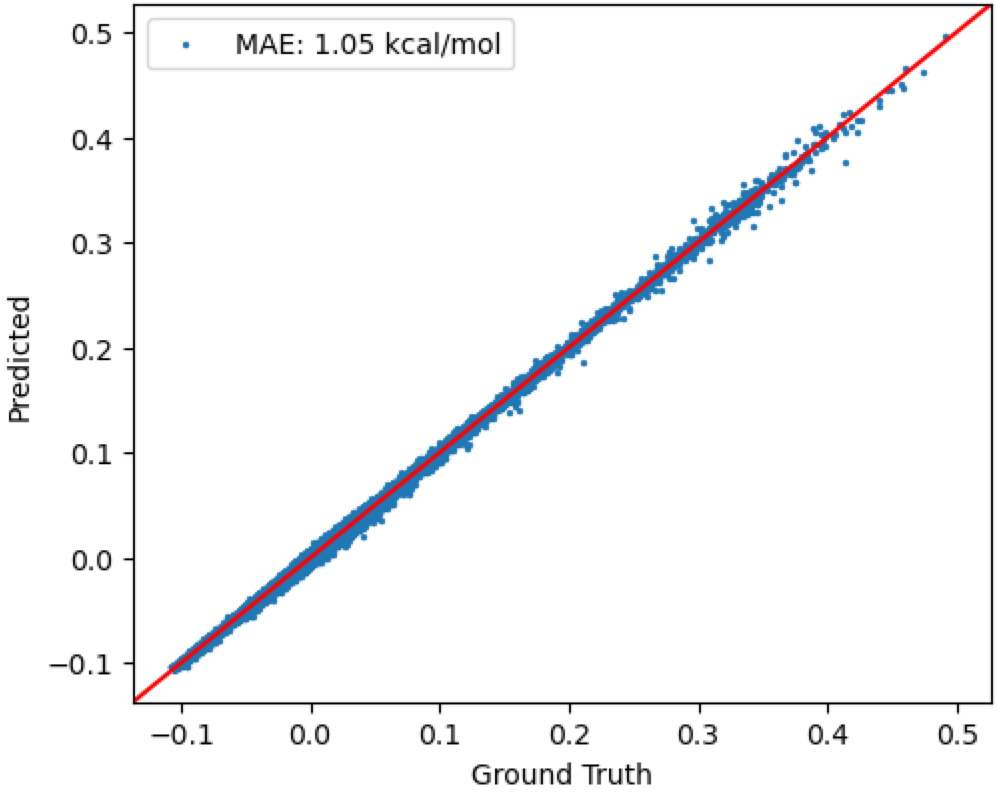


In [11]:
```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 288),
            nn.ReLU(),
            nn.Linear(288, 192),
            nn.ReLU(),
            nn.Linear(192, 96),
            nn.ReLU(),
            nn.Linear(96, 1)
        )

    def forward(self, x):
        return self.layers(x)

net_H = AtomicNet()
net_C = AtomicNet()
net_N = AtomicNet()
net_O = AtomicNet()

ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-3
num_epochs = 30
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=Fals
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 739972
Initialize training data...
```

```
100%|██████████| 30/30 [06:45<00:00, 13.51s/it]
```



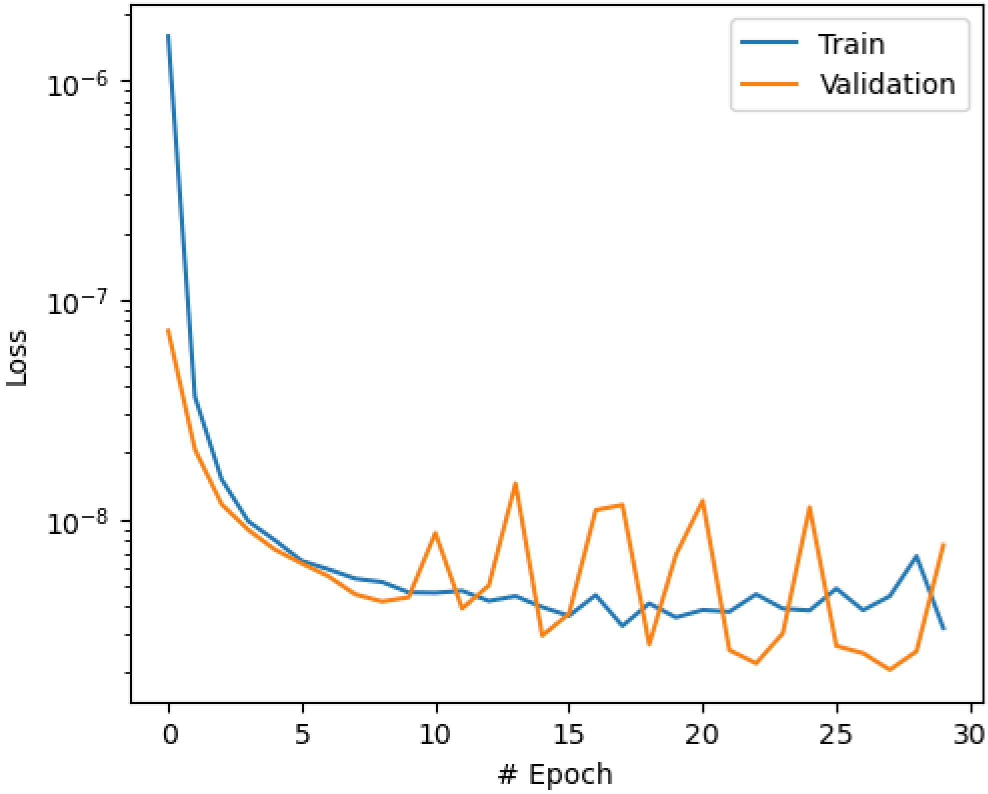


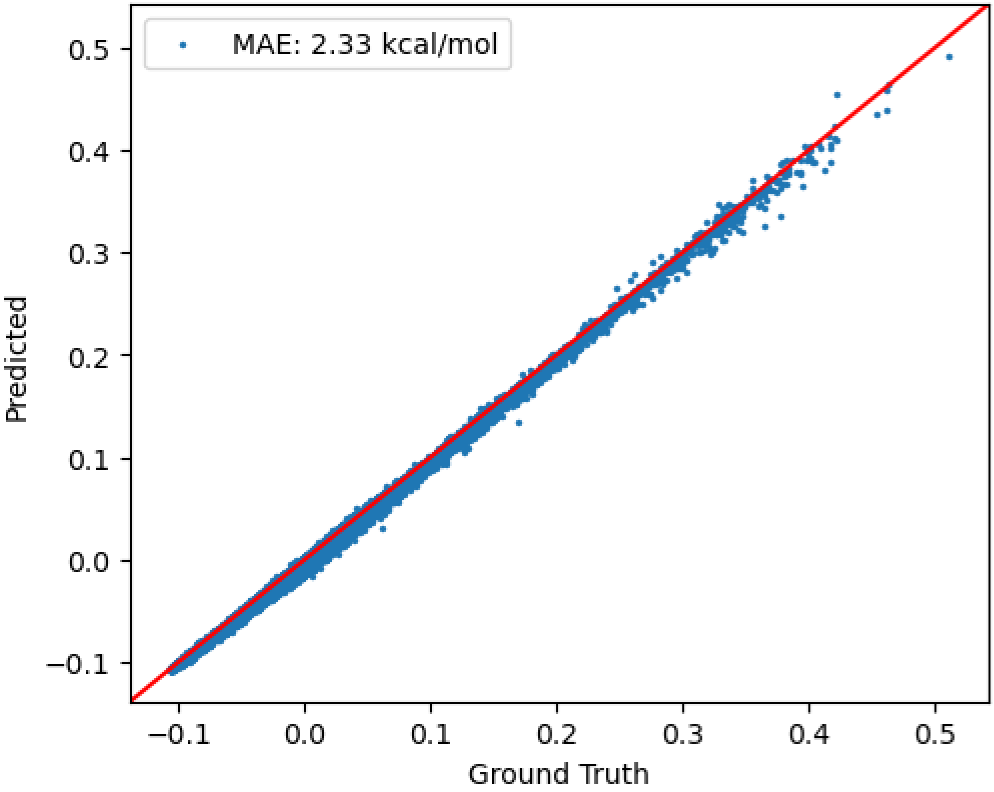
In [12]:

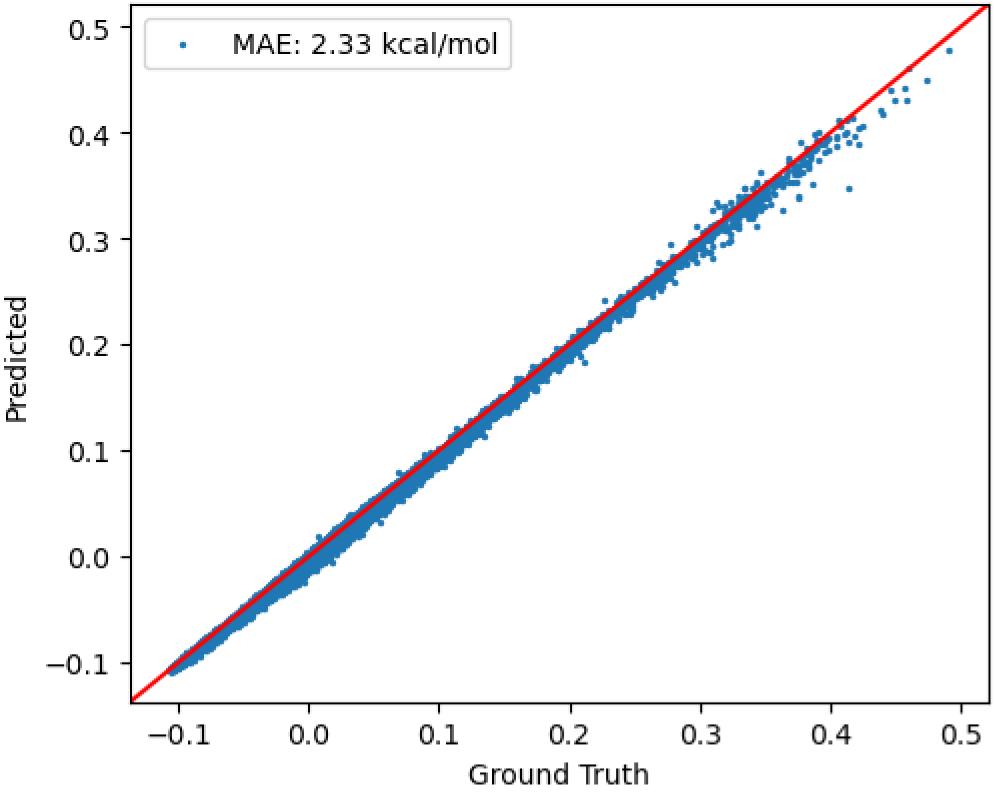
```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        return self.layers(x)

net_H = AtomicNet()
net_C = AtomicNet()
net_N = AtomicNet()
net_O = AtomicNet()

ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-3
num_epochs = 30
l2 = 0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=Fals
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 526340
Initialize training data...
```

```
100%|██████████| 30/30 [05:40<00:00, 11.36s/it]
```







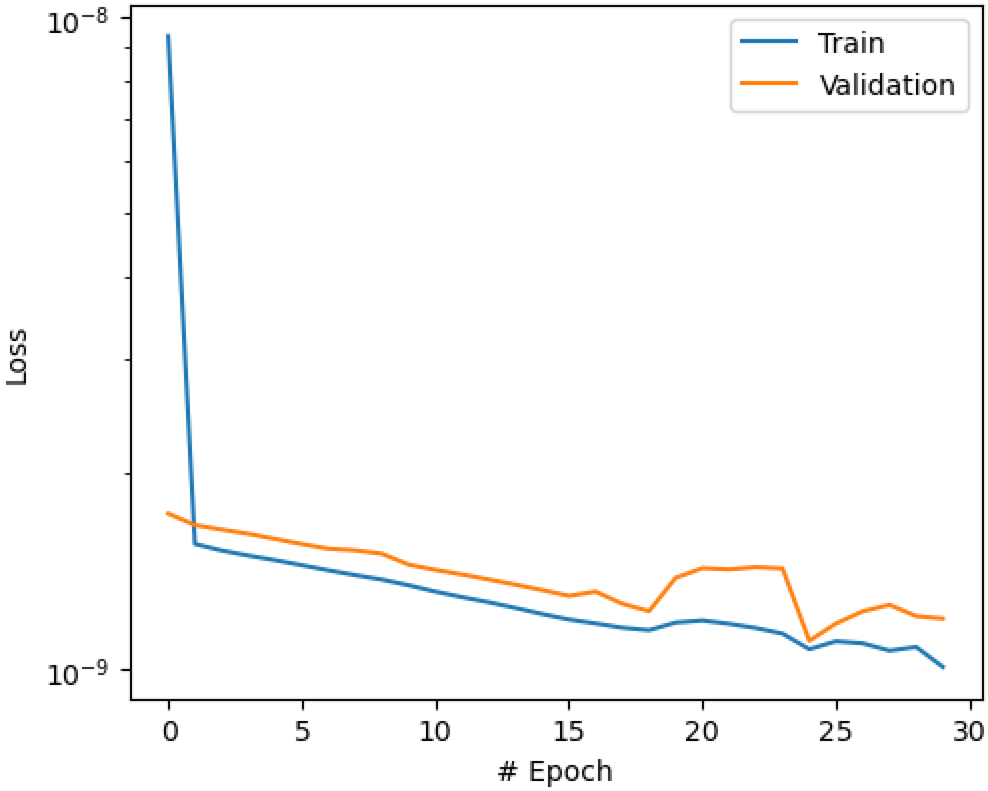
In [15]:

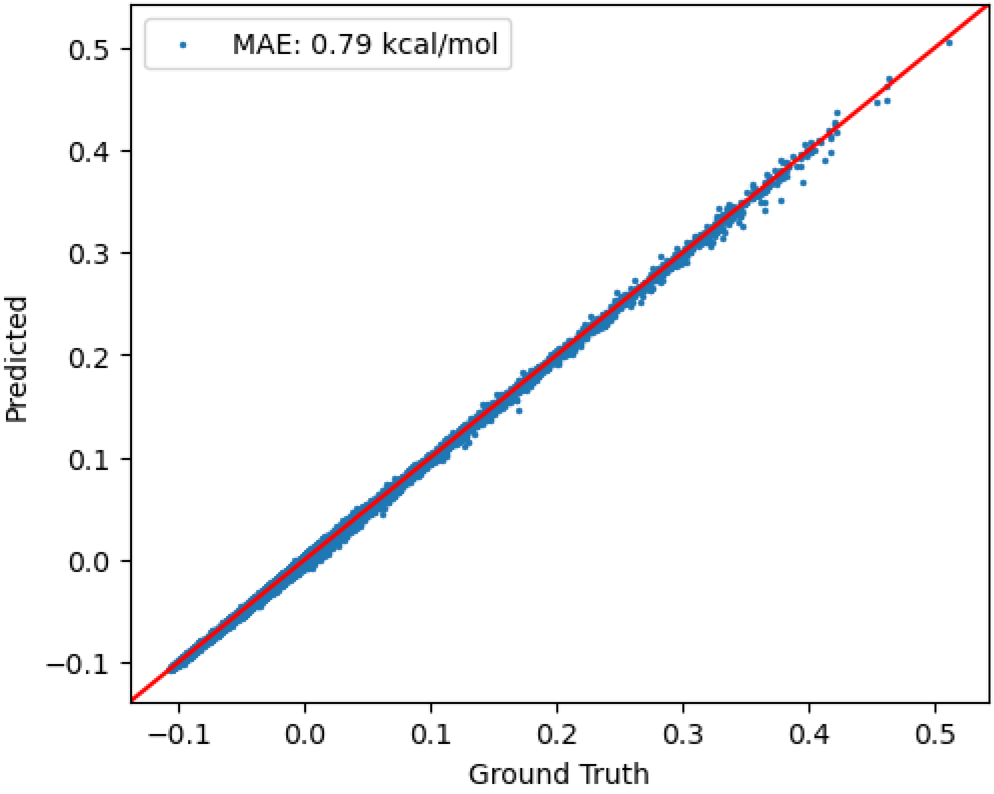
```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )
    def forward(self, x):
        return self.layers(x)
```

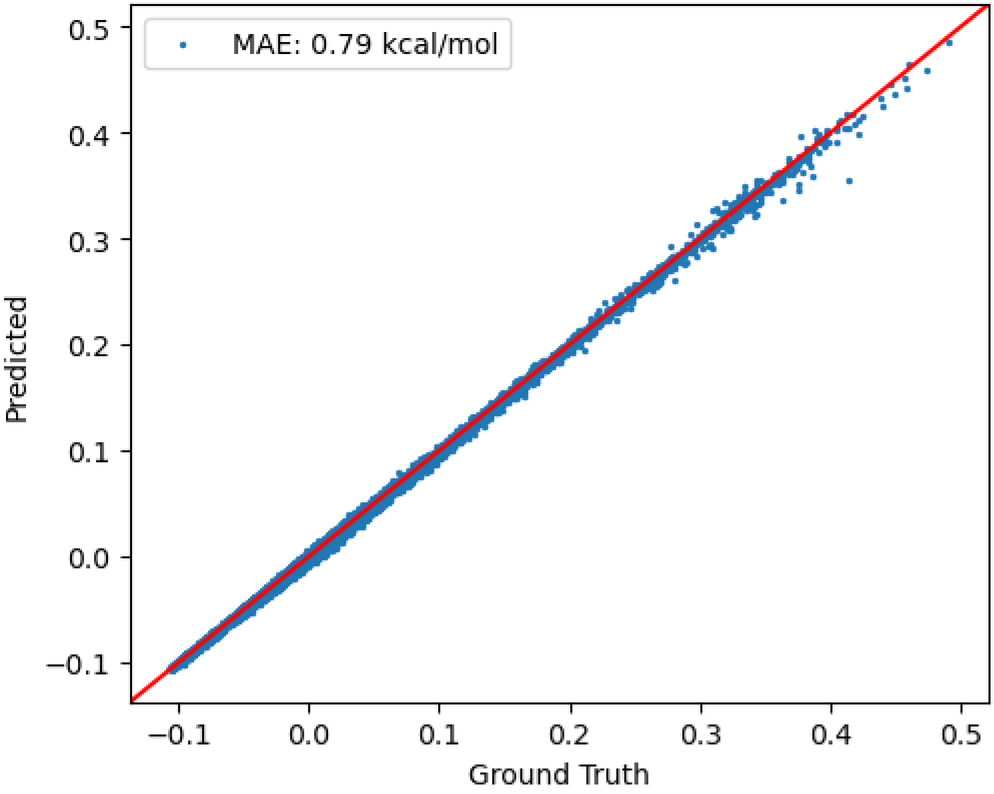
In [16]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 30
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 526340
Initialize training data...
```

```
100%|██████████| 30/30 [05:37<00:00, 11.26s/it]
```







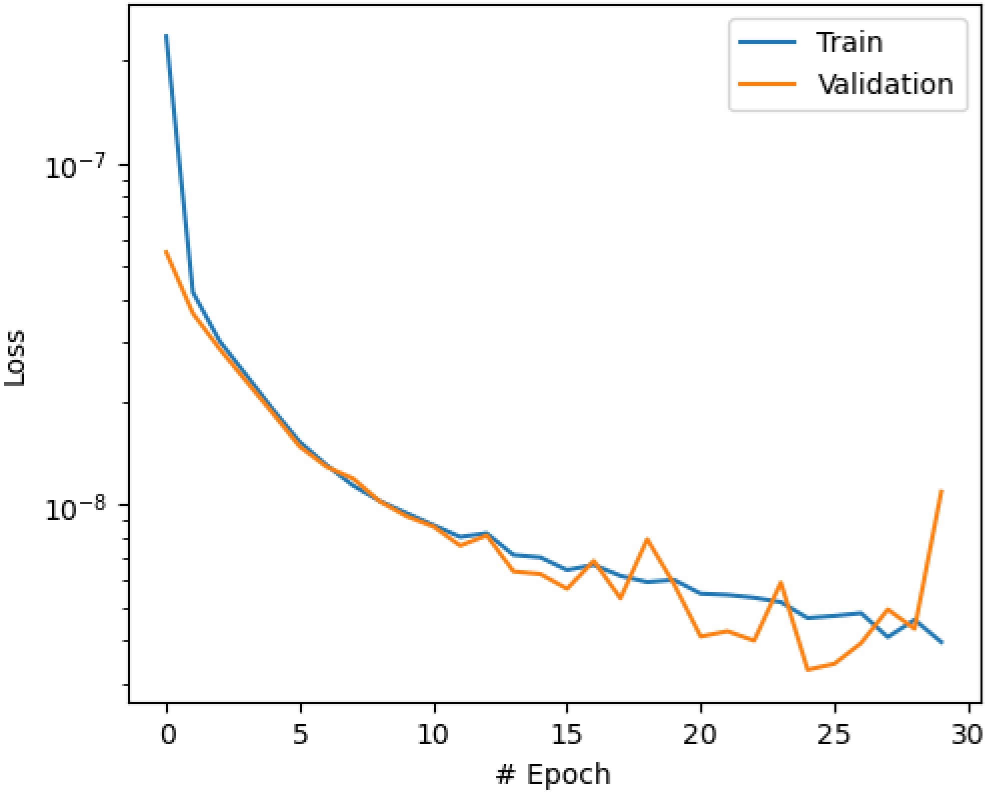
In [ ]:

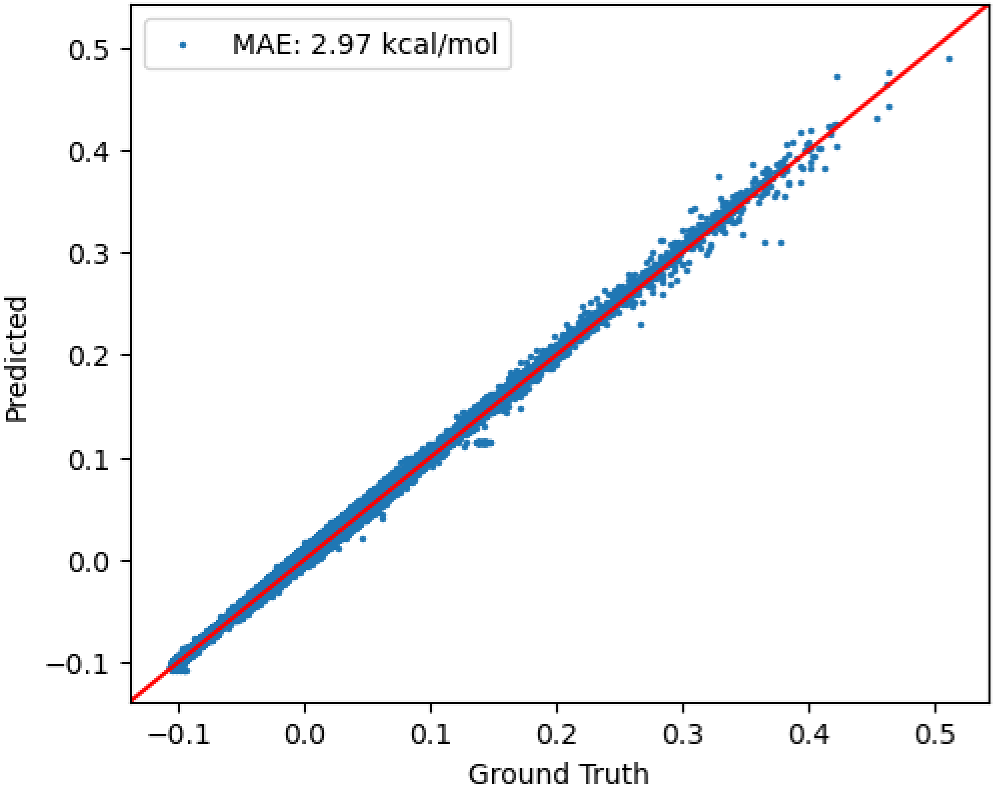
```
```

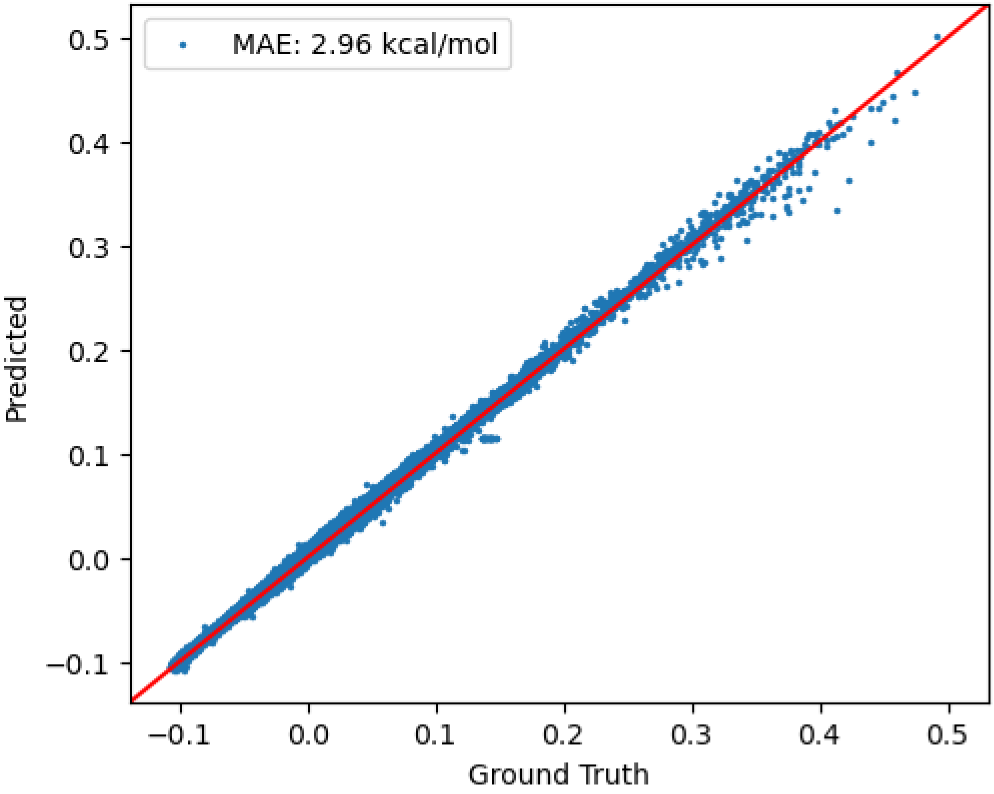
In [18]:

```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        return self.layers(x)

ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-3
num_epochs = 30
l2 = 0.0
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 526340
Initialize training data...
100%|██████████| 30/30 [05:50<00:00, 11.68s/it]
```



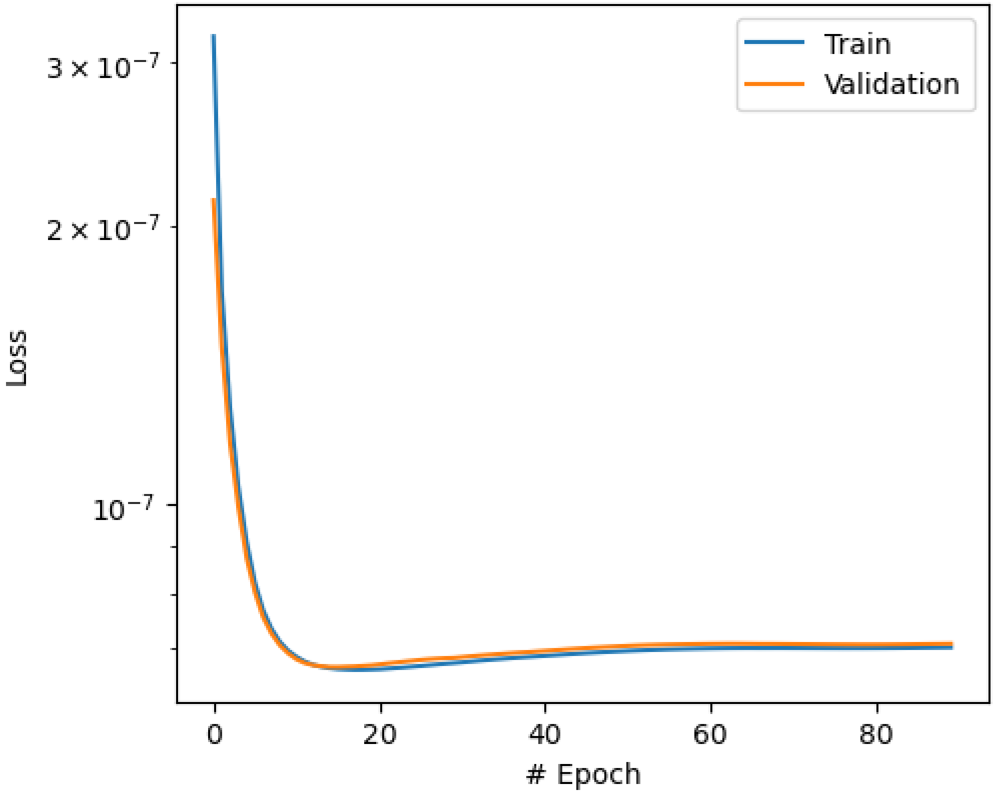


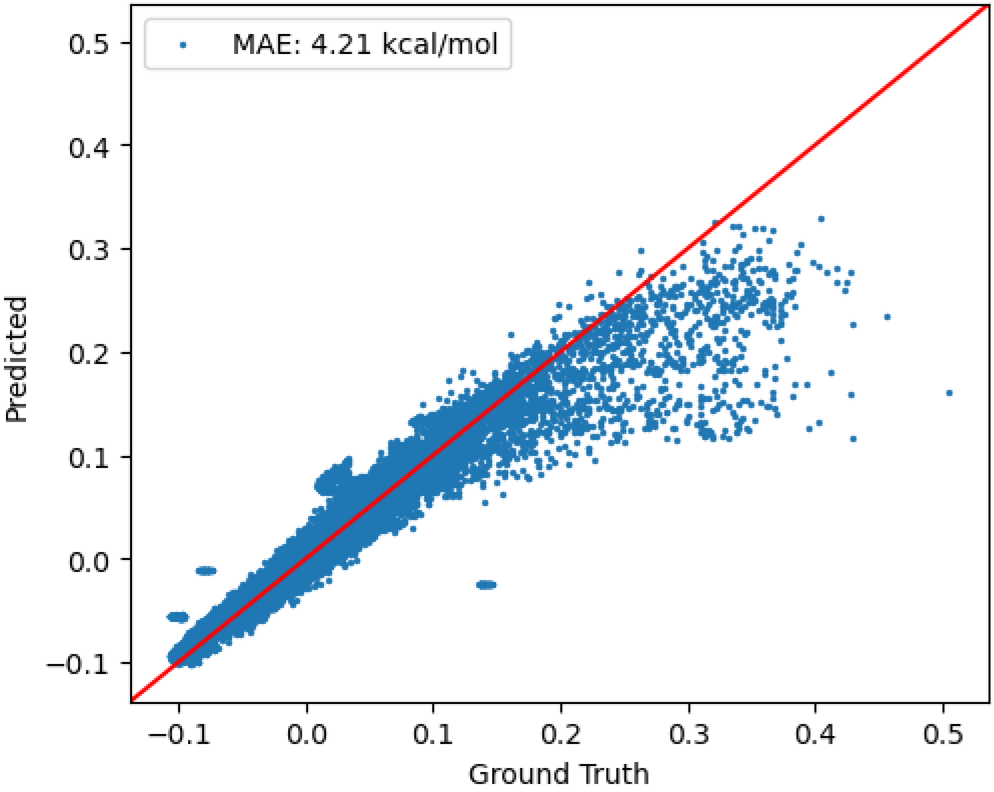


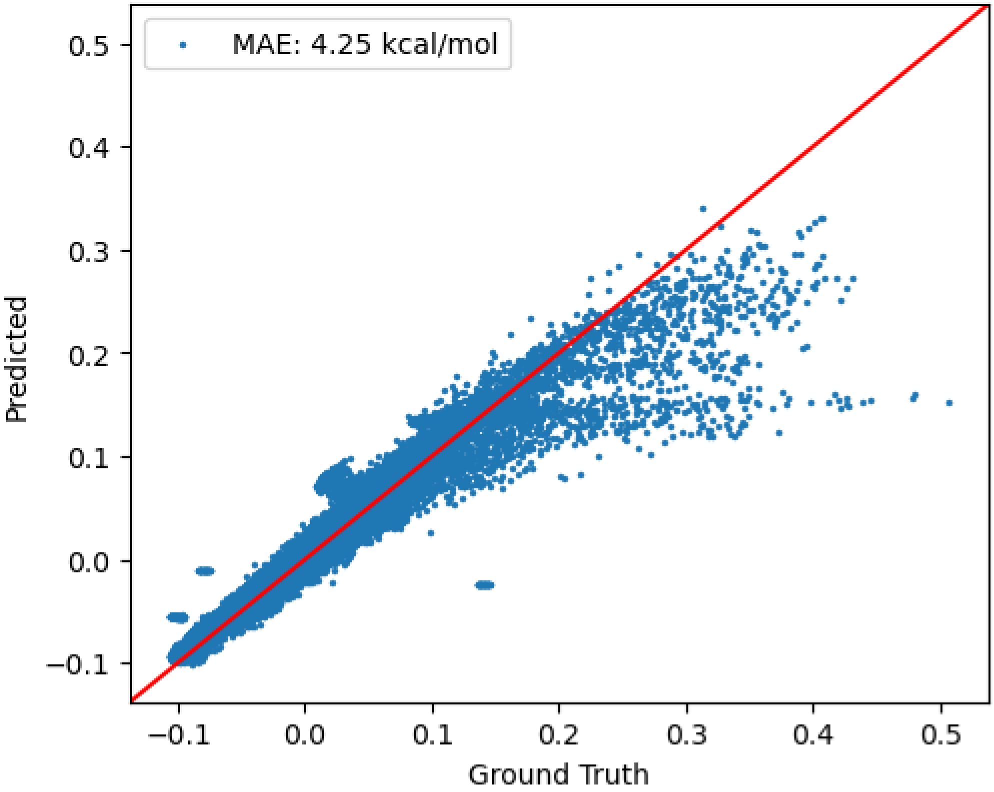
In [8]:
```
ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 90
```

```
l2 = 1e-3
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 90/90 [14:44<00:00,  9.83s/it]
```

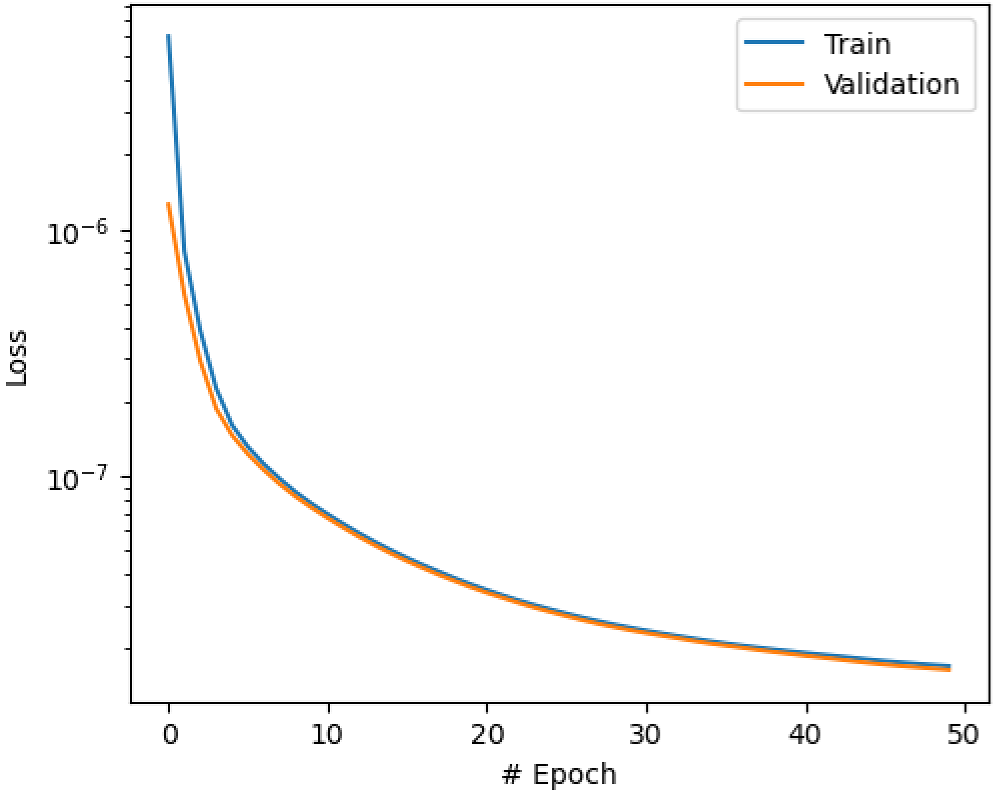


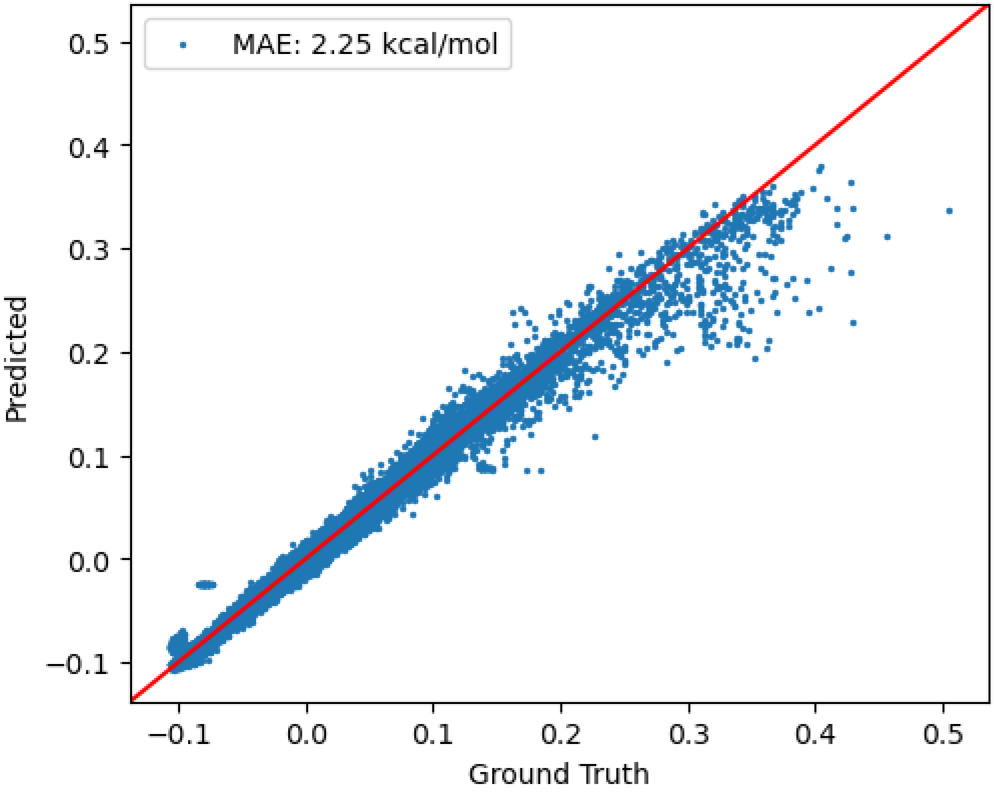


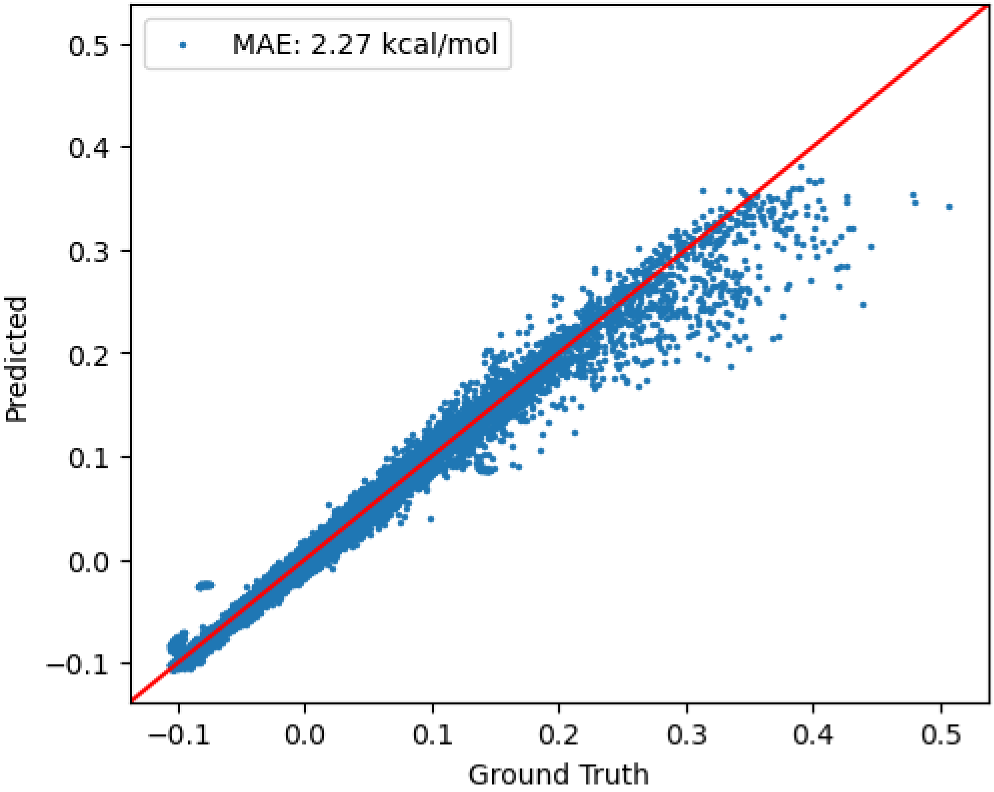


In [9]:
```
class AtomicNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(384, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )
    def forward(self, x):
        return self.layers(x)

net_H = AtomicNet()
net_C = AtomicNet()
net_N = AtomicNet()
net_O = AtomicNet()

ani_net = torchani.ANIModel([net_H, net_C, net_N, net_O]).to(device)
model = nn.Sequential(
    aev_computer,
    ani_net
).to(device)

learning_rate = 1e-4
num_epochs = 50
l2 = 1e-4
batch_size = 8162

trainer = ANITrainer(model, batch_size, learning_rate, num_epochs, l2)
train_losses, val_losses = trainer.train(train_data, val_data, early_stop=True
mae_val = trainer.evaluate(val_data)
mae_test = trainer.evaluate(test_data)
```

```
Sequential - Number of parameters: 197636
Initialize training data...
100%|██████████| 50/50 [07:50<00:00,  9.42s/it]
```




MAE: 2.25 kcal/mol
Predicted
Ground Truth
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.1
0.0
0.1
0.2
0.3
0.4
0.5


MAE: 2.27 kcal/mol
Predicted
Ground Truth
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.1
0.0
0.1
0.2
0.3
0.4
0.5