Life Science Model 1: To Predict New Molecules:

MoLFormer-XL-both-10%

MoLFormer is a class of models pretrained on SMILES string representations of up to 1.1B molecules from ZINC and PubChem. This repository is for the model pretrained on 10% of both datasets.

Dataset Size:

```
Train set size: (12000, 13)
Test set size: (5896, 2)
```

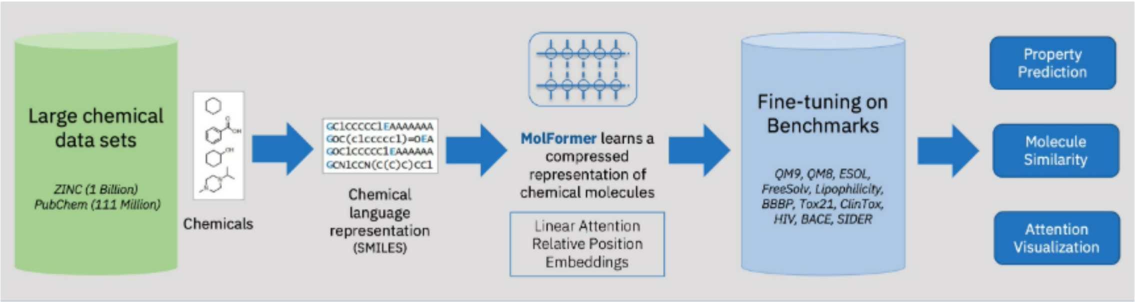
Sample Input:

```
Sample rows from training data:
   Unnamed: 0                                             smiles  task1
0           0  CC=C(CCC(C)C1CCC2C3=CCC4C(C)C(O)CCC4(C)C3CCC21...      0
1           1                 O=c1cc(-c2ccccc2)oc2cc(O)cc(O)c12      0
2           2                                  CC(C)(C)OCC1CO1      0
3           3         CN(C(=O)CCCOc1ccc2[nH]c(=O)ccc2c1)C1CCCCC1      0
4           4       C(=C/c1ccccc1)\CN1CCN(C(c2ccccc2)c2ccccc2)CC1      0
```

Component	Description
Model Type	Transformer-XL variant (MoLFormer-XL)
Task	Multi-task binary classification
Input	SMILES strings (molecular representations)
Tokenizer	Learned substructure-aware tokenizer
Architecture	Embedding → Transformer layers → [CLS]/pool → Linear
Loss Function	BCE With Logits Loss
Activation	Sigmoid on logits

Model Description

MoLFormer is a large-scale chemical language model designed with the intention of learning a model trained on small molecules which are represented as SMILES strings. MoLFormer leverges masked language modeling and employs a linear attention Transformer combined with rotary embeddings.



MoLFormer-XL-both-10%

MoLFormer is a class of models pretrained on SMILES string representations of up to 1.1B molecules from ZINC and PubChem. This repository is for the model pretrained on 10% of both datasets.

- **Ref:** https://huggingface.co/ibm-research/MoLFormer-XL-both-10pct

Steps:

- **Load & Preprocess Data**
  - Read SMILES strings and task labels from CSV files (train and test files).

- **Tokenizer Initialization**
  - Use the pretrained MoLFormer-XL tokenizer to convert SMILES strings into token IDs.
- **Model Architecture Setup**
  - Load MoLFormer-XL with a classification head (num_labels=1) for binary classification.
  - It adds a linear layer on top of a deep Transformer-XL encoder.
- **Custom Dataset Definition**
  - Create a SMILESDataset class to return tokenized inputs (and labels during training).

- **Task-wise Looping**
  - For each of the 11 tasks (task1 to task11), repeat fine-tuning and inference separately.
- **Tokenization of Inputs**
  - Tokenize both training and test SMILES strings for the current task using the same tokenizer.
  - Pad and truncate to fixed size for uniform input.
- **Dataloader Preparation**
  - Create PyTorch DataLoader objects for both training and test sets, enabling batching and shuffling.
- **Model Training**
  - Fine-tune the MoLFormer model using 8 epochs of training on the filtered dataset.
  - Use BCEWithLogitsLoss and AdamW optimizer to update weights.
- **Prediction (Inference)**
  - Pass test SMILES through the trained model, apply sigmoid() to logits to get probabilities.

```
[1] Input SMILES: ['N=c1nc(O)c2ncn(COC(CO)CO)c2[nH]1', 'CC[C@]1(O)C[C@H]2CN(CCc3c([nH]c4ccccc34)[C@@](C(=O)OC)(c3cc4c(cc3OC)
[2] Token IDs:
tensor([[ 0, 10, 12,  4,  6,  9,  7, 10, 26, 10, 12,  4, 26,  5,  8,  5,  5,  5,
          6, 30,  6, 12,  9,  7, 31,  7, 22,  8,  1,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2],
        [ 0,  9, 12,  4,  6,  9,  4,  8,  4,  4, 11,  4,  4,  4,  6,  4,  8,  7,
         30, 11,  8,  4,  4,  4,  4,  8,  7,  4,  6,  9,  7,  6,  5,  8,  5,  5,
          5,  5,  5,  8,  7,  5,  8,  5,  5,  5,  5,  5,  8,  1,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2]], device='cuda:0')
[3] Token Embeddings Shape: torch.Size([16, 242, 768])
[4] Last Hidden Layer Shape: torch.Size([16, 242, 768])
[5] Logits Output (pre-sigmoid): tensor([-0.3631,  0.0641], device='cuda:0')
[6] Sigmoid Probabilities: tensor([0.4102, 0.5160], device='cuda:0')
```

Architecture Core components:

1.Embedding Layer:

- Maps SMILES tokens to dense vectors (token_embeddings).
- Learned embeddings for atoms, bonds, special tokens (like = for double bond, etc.).

2.Transformer Layers (XL):

- Deep stack (24+ layers) with **self-attention** and **memory mechanism** from Transformer-XL.
- XL allows longer dependencies by caching past key-value states, useful for long SMILES strings.

3.Sequence Pooling:

- Uses [CLS] token or mean pooling over the sequence output.
- Produces a **fixed-size vector** for the full SMILES string.

4.Classification Head:

- **Linear projection layer**: [hidden_dim] → [1] (scalar logit per sample).
- Logits later passed through sigmoid() to get probabilities.

```
[1] Input SMILES: ['N=c1nc(O)c2ncn(COC(CO)CO)c2[nH]1', 'CC[C@]1(O)C[C@H]2CN(CCc3c([nH]c4ccccc34)[C@@](C(=O)OC)(c3cc4c(cc3OC)
[2] Token IDs:
tensor([[ 0, 10, 12,  4,  6,  9,  7, 10, 26, 10, 12,  4, 26,  5,  8,  5,  5,  5,
          6, 30,  6, 12,  9,  7, 31,  7, 22,  8,  1,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2],
        [ 0,  9, 12,  4,  6,  9,  4,  8,  4,  4, 11,  4,  4,  4,  6,  4,  8,  7,
         30, 11,  8,  4,  4,  4,  4,  8,  7,  4,  6,  9,  7,  6,  5,  8,  5,  5,
          5,  5,  5,  8,  7,  5,  8,  5,  5,  5,  5,  5,  8,  1,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
          2,  2,  2,  2,  2,  2,  2,  2]], device='cuda:0')
[3] Token Embeddings Shape: torch.Size([16, 242, 768])
[4] Last Hidden Layer Shape: torch.Size([16, 242, 768])
[5] Logits Output (pre-sigmoid): tensor([-0.3631,  0.0641], device='cuda:0')
[6] Sigmoid Probabilities: tensor([0.4102, 0.5160], device='cuda:0')
```

While During Training:

```
Train set size: (12000, 13)
Test set size: (5896, 2)

Sample rows from training data:
   Unnamed: 0                                             smiles  task1  \
0           0  CC=C(CCC(C)C1CCC2C3=CCC4C(C)C(O)CCC4(C)C3CCC21...      0
1           1                  O=c1cc(-c2ccccc2)oc2cc(O)cc(O)c12      0
2           2                                    CC(C)(C)OCC1CO1      0
3           3         CN(C(=O)CCCOc1ccc2[nH]c(=O)ccc2c1)C1CCCCC1      0
4           4      C(=C/c1ccccc1)\CN1CCN(C(c2ccccc2)c2ccccc2)CC1      0

   task2  task3  task4  task5  task6  task7  task8  task9  task10  task11
0      0      0      0     -1     -1      0      0      0       0       0
1      0      0      0      1      0      0      0      0       0      -1
2      0      0      0     -1     -1      0      0      0       0       0
3      0      0      0     -1     -1      0      0      0       0      -1
4      0      0      0      0      0     -1      0      0       0      -1

Sample rows from test data:
   Unnamed: 0                                             smiles
0           0      Cc1ncc(CN2CC=C(c3ccccc3)CC2)c(=N)[nH]1.Cl.Cl
1           1                               N=C(N)Nc1ccc(Cl)cc1
2           2  NCC1OC(OC2C(N)CC(N)C(OC3OC(CO)C(O)C(N)C3O)C2O)...
3           3                                        ClCC(Br)CBr
4           4                       Oc1ccc2[nH]cc(C3=CCNCC3)c2n1
```

```
Processing task2...
Some weights of Molform
You should probably TRA
Epoch 1 loss: 0.6916
Epoch 2 loss: 0.5955
Epoch 3 loss: 0.5117
Epoch 4 loss: 0.3973
Epoch 5 loss: 0.2945
Epoch 6 loss: 0.2298
Epoch 7 loss: 0.1446
Epoch 8 loss: 0.1652
```

```
Processing task1...
Some weights of Molformer
You should probably TRAIN
Epoch 1 loss: 0.2768
Epoch 2 loss: 0.1711
Epoch 3 loss: 0.1202
Epoch 4 loss: 0.0802
Epoch 5 loss: 0.0557
Epoch 6 loss: 0.0449
Epoch 7 loss: 0.0267
Epoch 8 loss: 0.0251
```

AUC mean	AUC Task1	AUC Task2	AUC Task3	AUC Task4	AUC Task5	AUC Task6	AUC Task7	AUC Task8	AUC Task9
0.702	0.875	0.615	0.905	0.333	0.877	0.696	0.359	0.654	0.888
0.652	0.892	0.505	0.922	0.69	0.673	0.49	0.434	0.667	0.9

Final AUC Results: