A. MODEL SUMMARY

A1. Background on you/your team

Competition Name: leash-BELKA
(https://www.kaggle.com/competitions/leash-BELKA)
Team Name: mamba1-one-fold-lb0.432
Private Leaderboard Score: 0.28557
Private Leaderboard Place: rank 5th

A2. Background on you/your team

What is your academic/professional background?
I am a contract computer vision and deep learning algorithm engineer. My job includes discovering fracture in x-ray images and implementing visual slam for robotic navigation. Recently, I am helping companies to finetune LLM models for their applications.

Did you have any prior experience that helped you succeed in this competition?
I am familiar deep learning and build deep models in my work. I also have experiences in making models for medicine and molecular applications from previous Kaggle competitions like Bristol-Myers Squibb – Molecular Translation, OpenVaccine: COVID-19 mRNA Vaccine Degradation Prediction.

What made you decide to enter this competition?
The problem of virtual screening in DNA encoded libraries is interesting. I hope can learn from other fellow kagglers about the latest AI methods and insights in this area.

How much time did you spend on the competition?
About 5 hours per day, over two months.

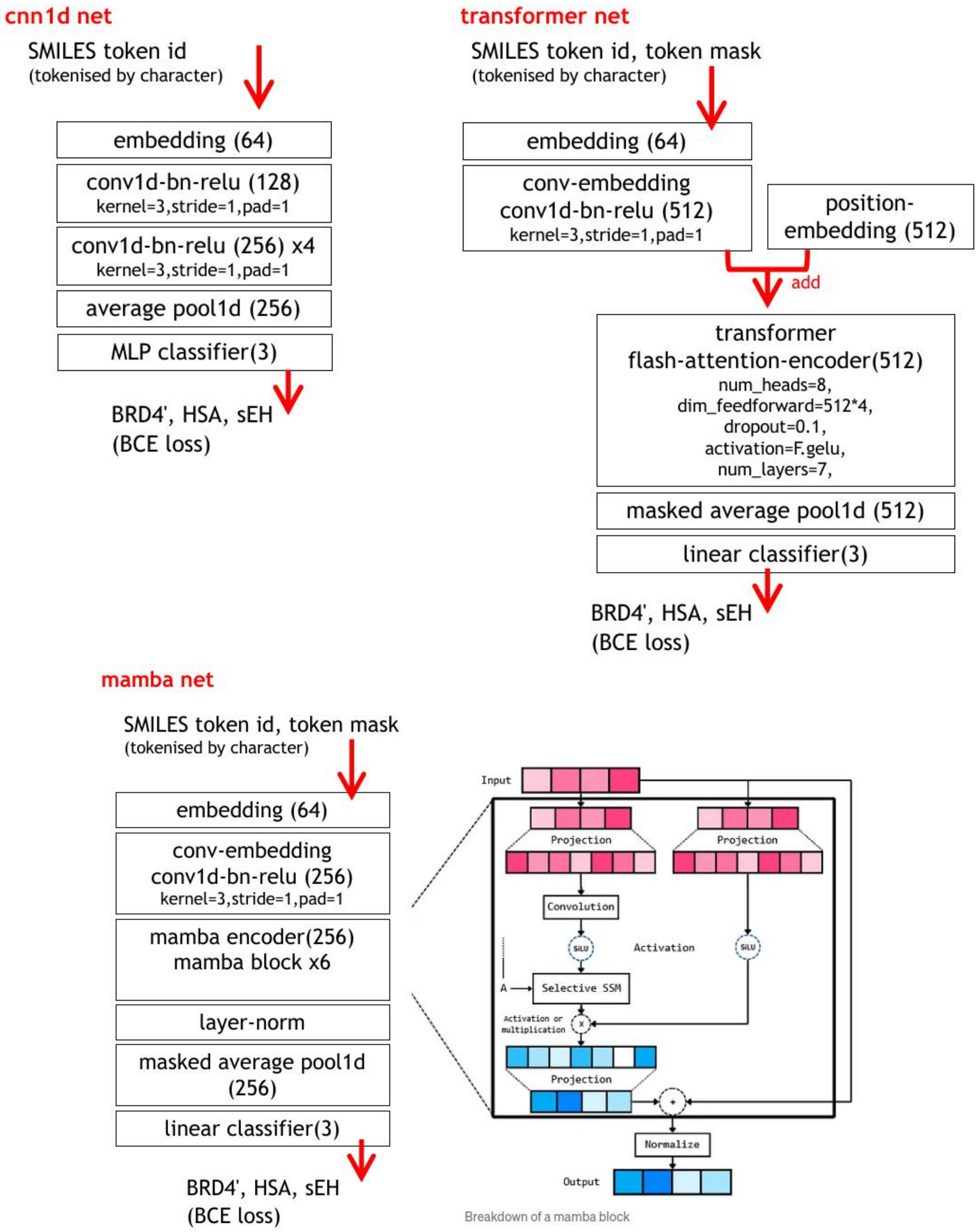
A3. Summary

In this competition, our task is to predict the binding affinity of small molecules to specific protein targets – a critical step in drug discovery. Since there are only 3 target proteins, we can treat this as a multi-label classification problem, i.e. bind versus non-bind for target protein, given an input molecule smiles string.

One issue in the kaggle competition is that a large part of the test molecules are synthesized from different building blocks used in the train molecules. To mitigate the effects of OOD (out-of-domain distribution), we use an ensemble of different sequence models. The solution ensemble consists of 3-fold conv1d, 2-fold transformer and single-fold mamba (SSM, selective state machines) nets, as shown in Figure.1.

All nets are build with python and pytorch deep learning framework. In order to handle 98 millions of training molecules efficiently, we use customized cuda kernel from open source flash attention[1] and mamba SSM[2] libraries.

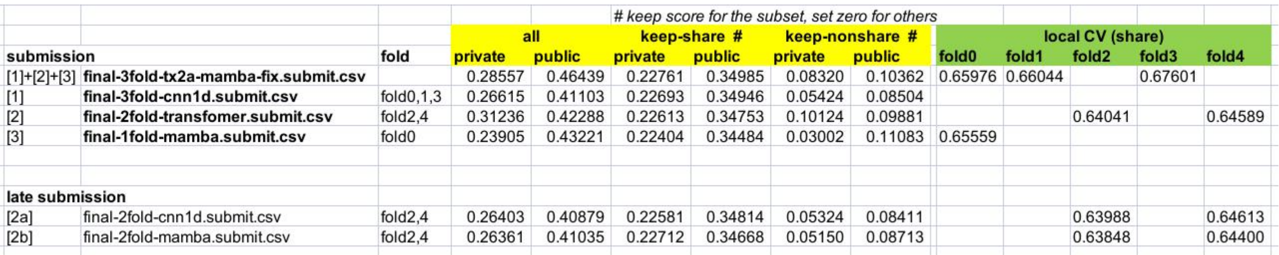
For training we have two Nvidia Ada A6000 / 48 GB Ampere GPUs. Time for training one fold with single GPU are: 7 hr for cnn1d, 28 hr for transformer, and 36 hr for mamba,



Breakdown of a mamba block

Figure.1 : Different nets used in ensemble solution

Figure.2 shows the performances of various nets and the ensemble solution on the private and public leader board. We also make some late submissions (i.e. submission after the competition) to further analyse the models.



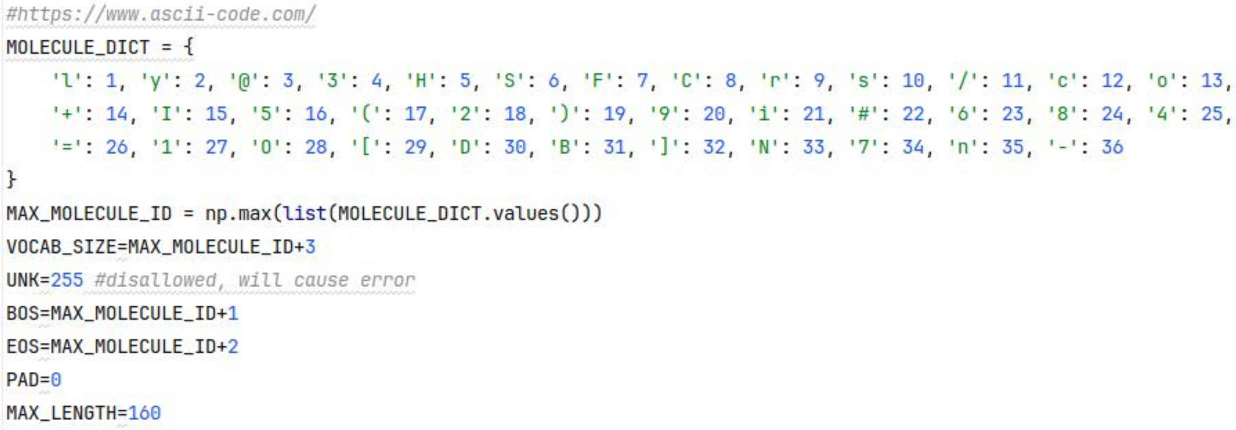
					# keep score for the subset, set zero for others								
			all		keep-share #		keep-nonshare #		local CV (share)				
submission		fold	private	public	private	public	private	public	fold0	fold1	fold2	fold3	fold4
[1]+[2]+[3]	**final-3fold-tx2a-mamba-fix.submit.csv**		0.28557	0.46439	0.22761	0.34985	0.08320	0.10362	0.65976	0.66044		0.67601	
[1]	**final-3fold-cnn1d.submit.csv**	fold0,1,3	0.26615	0.41103	0.22693	0.34946	0.05424	0.08504					
[2]	**final-2fold-transfomer.submit.csv**	fold2,4	0.31236	0.42288	0.22613	0.34753	0.10124	0.09881			0.64041		0.64589
[3]	**final-1fold-mamba.submit.csv**	fold0	0.23905	0.43221	0.22404	0.34484	0.03002	0.11083	0.65559				
late submission													
[2a]	final-2fold-cnn1d.submit.csv	fold2,4	0.26403	0.40879	0.22581	0.34814	0.05324	0.08411			0.63988		0.64613
[2b]	final-2fold-mamba.submit.csv	fold2,4	0.26361	0.41035	0.22712	0.34668	0.05150	0.08713			0.63848		0.64400

Figure.2 : Performance of different nets in the leader board

A4. Features Selection / Engineering

1. Tokenization

We need convert SMILES string into tokens as input to our sequence models. We tried several methods like character based, sentence piece, byte-pair-encoding (BPE), atom/smiles notation aware to break the SMILES strings. Surprisingly, the simplest character based tokenization performs the best across different net architecture, see Figure.3.



```
#https://www.ascii-code.com/
MOLECULE_DICT = {
    'l': 1, 'y': 2, '@': 3, '3': 4, 'H': 5, 'S': 6, 'F': 7, 'C': 8, 'r': 9, 's': 10, '/': 11, 'c': 12, 'o': 13,
    '+': 14, 'I': 15, '5': 16, '(': 17, '2': 18, ')': 19, '9': 20, 'i': 21, '#': 22, '6': 23, '8': 24, '4': 25,
    '=': 26, '1': 27, 'O': 28, '[': 29, 'D': 30, 'B': 31, ']': 32, 'N': 33, '7': 34, 'n': 35, '-': 36
}
MAX_MOLECULE_ID = np.max(list(MOLECULE_DICT.values()))
VOCAB_SIZE=MAX_MOLECULE_ID+3
UNK=255 #disallowed, will cause error
BOS=MAX_MOLECULE_ID+1
EOS=MAX_MOLECULE_ID+2
PAD=0
MAX_LENGTH=160
```

Figure.3 : character-based tokenization

We further add a conv1d layer of kernel size=3, stride=1 to learned combinations of consecutive tokens (bi-grams, tri-grams) before passing them into cnn1d, transformer or mamba encoder.

2. Batch normalization

We find that the model performance is sensitive to batch normalization. We think this is because:
- in-distribution and out-distribution samples have different feature values.
- class is imbalance (positive class is less than 1%). Note that positive and negative samples also have different feature values.

To alleviate the problem, we use high eps=5e-3 and low momentum=0.2 for cnn1d net.

A5. Training Method(s)

To train the sequence models, we use binary cross entropy loss. We perform back propagation with ADAM optimizer. The important hyper-parameters are:

- step learning rate of 1e-3,1e-4,1e-5 for 6-12 epochs.
- large batch size of 2000,2500,5000

Interesting, the best way to handle class imbalance is to do nothing (no up-sampling or under-sampling of the class). We think this could be because of the large batch size we used.

A6. Interesting findings

What was the most important trick you used?
From Figure.2, transformer has exceptional performance in the leader board scores. We think the use of character-based tokenization and conv embedding to learn meaningful bi-grams, tri-grams are important tricks to make transformer robust to OOD.

We perform more in depth comparison of the predictions made by cnn1d and transformer net. Figure.4 shows their correlation on validation set. Next, we generate the net prediction heatmap using GradCAM[3] and visualize it with XSMILES[4]. Figure.5 shows some heatmaps. As expected, cnn1d has local activations, whereas transformer has more global ones.

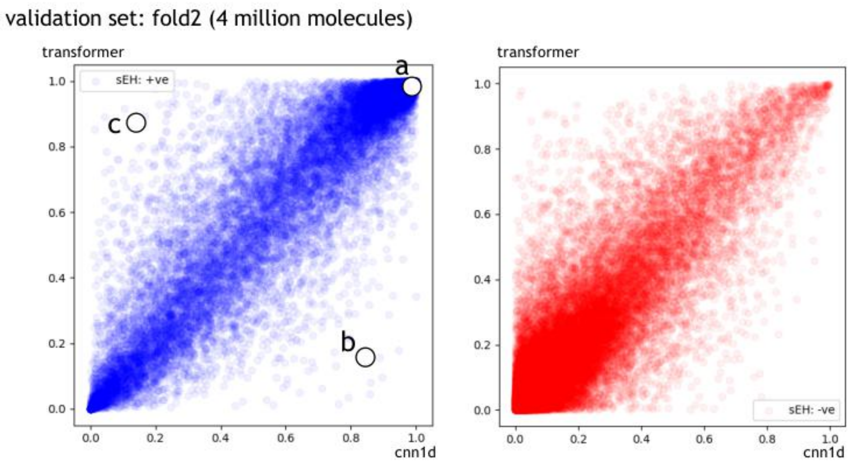
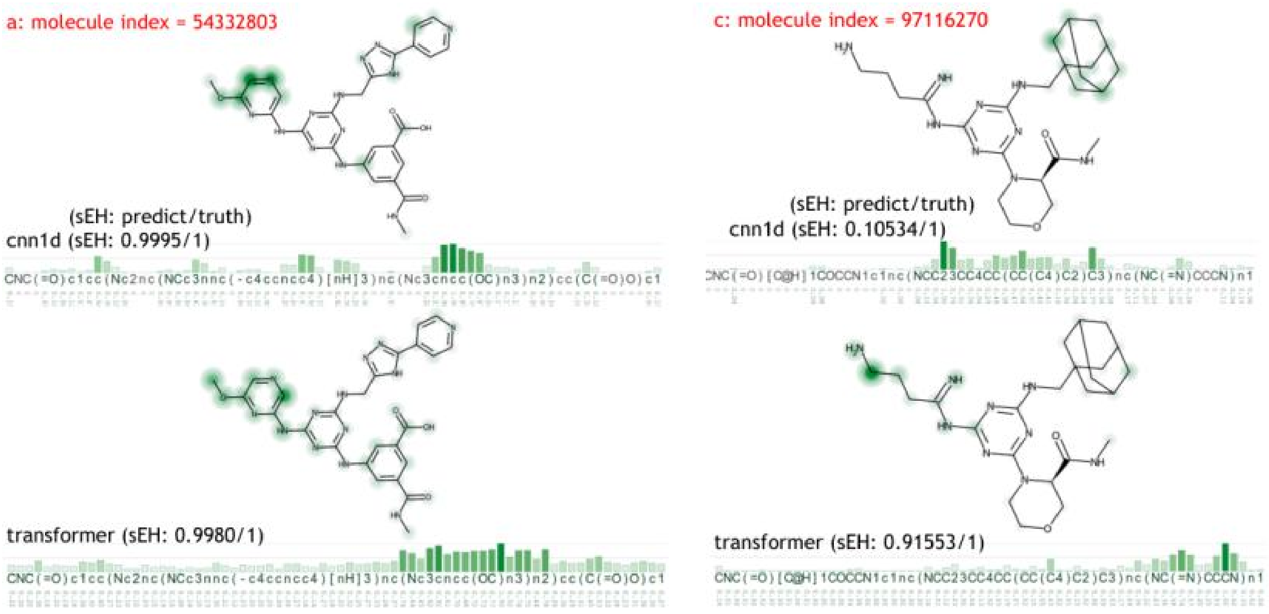


Figure.4 : Correlations of cnn1d and transformer predictions


a: molecule index = 54332803
(sEH: predict/truth)
cnn1d (sEH: 0.9995/1)
CNC(=O)c1cc(Nc2nc(NCc3nnc(-c4ccncc4)[nH]3)nc(Nc3cncc(OC)n3)n2)cc(C(=O)O)c1
transformer (sEH: 0.9980/1)
CNC(=O)c1cc(Nc2nc(NCc3nnc(-c4ccncc4)[nH]3)nc(Nc3cncc(OC)n3)n2)cc(C(=O)O)c1
c: molecule index = 97116270
(sEH: predict/truth)
cnn1d (sEH: 0.10534/1)
CNC(=O)[C@H]1COCCN1c1nc(NCC23CC4CC(CC(C4)C2)C3)nc(NC(=N)CCCN)n1
transformer (sEH: 0.91553/1)
CNC(=O)[C@H]1COCCN1c1nc(NCC23CC4CC(CC(C4)C2)C3)nc(NC(=N)CCCN)n1

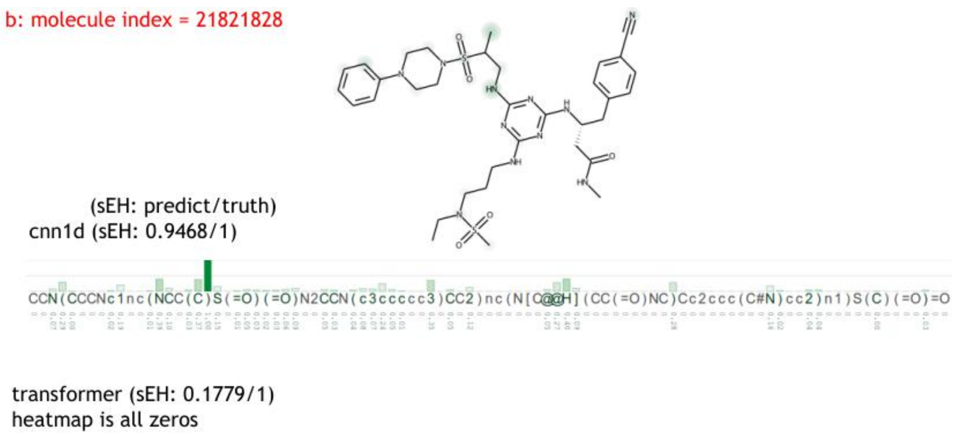

b: molecule index = 21821828
(sEH: predict/truth)
cnn1d (sEH: 0.9468/1)
CCN(CCCNc1nc(NCC(C)S(=O)(=O)N2CCN(c3ccccc3)CC2)nc(N[C@@H](CC(=O)NC)Cc2ccc(C#N)cc2)n1)S(C)(=O)=O
transformer (sEH: 0.1779/1)
heatmap is all zeros

Figure.4 : Heatmaps of cnn1d and transformer predictions

What do you think set you apart from others in the competition?
We have very powerful GPU cards. The two Nvidia Ada A6000 / 48GB Ampere GPUs make our training fast. To use large batch size of 2500 in transformer, we spend efforts to write memory-efficient code. In particular, we have to discard pytorch dataloader that can led to growing memory usage in cpu RAM. This is because the loader uses python multi-process which copy the batch to queue and memory is not released until the whole epoch ends.

A7. Simple Features and Methods

Many customers are happy to trade off model performance for simplicity. With this in mind:
Is there a subset of features that would get 90-95% of your final performance?

From Figure.2, it is sufficient just to use transformer net.

A8. Model Execution Time

How long does it take to train your model?
Time for training one fold with single GPU are: 7 hr for cnn1d, 28 hr for transformer, and 36 hr for mamba.

How long does it take to generate predictions using your model?
The test data has about 880,000 molecules. It take less than 1 minutes for each net to process them.

How long does it take to train the simplified model (referenced in section A7)?
I expect a reduction of more than 50% in time.

How long does it take to generate predictions from the simplified model?
I expect a reduction of more than 50% in time.

A9. References

[1] “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness” - Tri Dao
https://github.com/Dao-AILab/flash-attention

[2] “Mamba: Linear-Time Sequence Modeling with Selective State Spaces” - Albert Gu, Tri Dao
https://github.com/state-spaces/mamba

[3] Advanced AI explainability for PyTorch
https://github.com/jacobgil/pytorch-grad-cam

[4] Molecular structures and SMILES visualization
https://github.com/Bayer-Group/xsmiles