ALTeGraD 2023 Data Challenge

Molecule Retrieval with Natural Language Queries

Hugo Queniat
Télécom Paris
hugo.queniat@telecom-paris.fr

Simon Queric
Télécom Paris
simon.queric@telecom-paris.fr

Abstract

The purpose of the data challenge *Molecule Retrieval with Natural Language Queries* was to find the best Machine Learning model that matches a description of chemical properties of a molecule with the right molecule in a given dataset.

1 Introduction

Molecules and texts are data with intrinsically different structures : molecules can be represented as graphs which are combinatorial objects, whereas a text is essentially a sequential data. As usual in Machine Learning, to overcome the difficulty of comparing different data structures, we are looking for vector representations of our molecules and text descriptions. It leads us to jointly train two Neural Networks : a Graph Neural Network (GNN) and a Text Encoder, that can extract information in vector form and output the so called graph and text embeddings. Once we have those vector representations, one can compute the similarity between the embeddings, with a distance chosen beforehand, allowing to map each representation from a model to the closest produced by its pair.

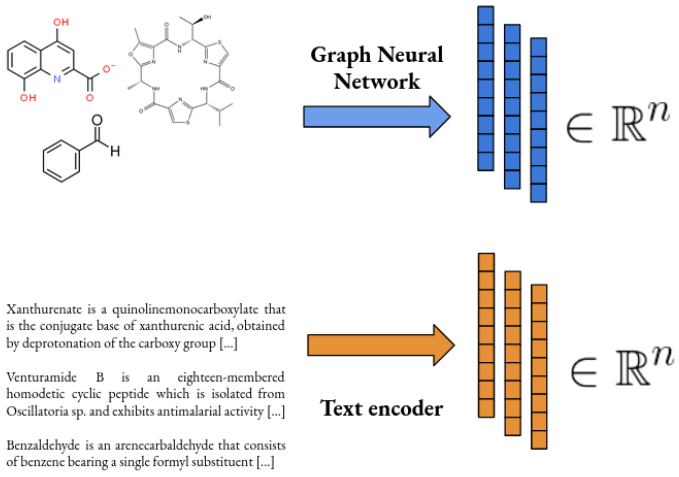


Figure 1: GNN and Text Encoder respectively map a graph and a text to a point in $\mathbb{R}^n$ where n is the embedding dimension.

Given a dataset of molecules, the goal is to retrieve the molecule from a textual description of its chemical properties. This is a real classification task, known as **Text2Mol** and first proposed in [4], where the model will output a vector of similarities for each molecule. This vector actually corresponds to the degree of similarity between each molecule in the set and the actual

molecule described in the text, according to the model.

As we stated earlier, to tackle this real classification task, we were tasked with building a bi-modal architecture in a contrastive learning setting. Indeed, the objective is to co-train both the Text Encoder and the Graph Encoder such that corresponding textual and molecular representations are mapped to close embeddings while representations that do not match are mapped to far vectors.

2 The Data Challenge

Before delving into the architecture design for our model, our initial focus was on comprehending the task at hand. Primarily, we sought to gain a deeper understanding of the dataset we would be working with and the scoring method employed to evaluate the model's performance.

The Dataset

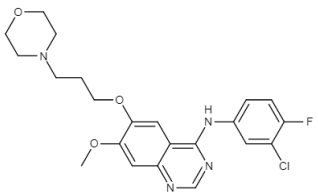
The data we were offered for this data challenge decomposed classically into 3 distinct sets :

- Training set with 26408 samples composed with a molecule ID and its paired textual description.
- Validation set with 3301 samples following the same decomposition.
- Test set with 3301 divided in two files, on one side textual descriptions and on the other molecule IDs.

The test set was not to be shuffled since we did not have any information on the correspondence between the textual descriptions and the molecule IDs. Indeed, the model is evaluated on its ability to draw this unknown correspondence.

Our very first observation about the dataset was on the cardinality of the sets. The validation set is about 12.5% the size of the training set which seems reasonable considering we have a training set of around 26k samples. Thus, we elected not to modify those sets.

Considering the elements of the sets, here is an instance of the training set :



Gefitinib is a member of the class of quinazolines that is quinazoline which is substituted by a (3-chloro-4-fluorophenyl)nitrilo group, 3-(morpholin-4-yl)propoxy group and a methoxy group at positions 4, 6, and 7, respectively. An EGFR kinase inhibitor used for the treatment of non-small cell lung cancer. It has a role as an epidermal growth factor receptor antagonist and an antineoplastic agent. It is an aromatic ether, a member of monochlorobenzenes, a member of monofluorobenzenes, a secondary amino compound, a tertiary amino compound, a member of quinazolines, and a member of morpholines.

Figure 2: A molecule identified with its **Molecular ID - 123631 -** against its textual description as given in the training dataset.

We observe that the textual description of the molecule in figure 2 is rather precise and uses specific jargon associated to chemistry, biology and molecular structures. This would lead us to search for a TextEncoder specifically tailored for the task of encoding such a vocabulary.

Another observation we made during our investigation, was the number of different molecules in the sets. Indeed, each molecular ID is present only once in the training set, meaning that for each molecule we have a single textual description to train the model with. This is a crucial observation because it means that for each molecule and textual description, we have a single positive sample. Consequently, this means that when building our loss metric, as we try to maximize the distance between an anchor and a negative sample and minimize the distance between an anchor and a positive sample, we cannot consider any other positive sample than our anchor.

This last observation limited massively our choice of metrics. Indeed, most known and recent metric losses for contrastive learning rely on the existence of several positive samples, such as the Triplet Loss [9] or the N-Pair Loss [11]. Nonetheless, we did try to implement those metrics in a hard-negative samples setting, meaning that we had only the anchor and negative samples. However, our experiments with those metrics turned inconclusive. As a result, we chose to stick to the more traditional contrastive loss [5].

The scoring method

The performance of our models is assessed using the label ranking average precision score :

$$\mathrm{LRAP}\left(y, y'\right) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{1}{\|y_i\|_0} \sum_{j:y_{ij}} \frac{|\mathcal{L}_{ij}|}{\mathrm{rank}_{ij}}$$

where $\mathcal{L}_{ij} = \left\{k : y_{ik} = 1, y'_{ik} \geq y'_{ij}\right\}, \mathrm{rank}_{ij} = \left|k : y'_{ik} \geq y'_{ij}\right|$.

Label ranking average precision (LRAP) is the average over each ground truth label assigned to each sample, of the ratio of true vs. total labels with lower score. If there is exactly one relevant label per sample - which is our case here -, label ranking average precision is equivalent to the mean reciprocal rank $\left(\mathrm{MRR}^2\right)$.

Studying in depth this metric of scoring, we came up with 2 conclusions for how we were supposed to build our architecture in order to achieve the best results :

- The probability distribution or output should not be deterministic as in "Dirac-like", meaning that if we were to predict such a distribution, if the prediction for a sample is not correct, then we only achieve an accuracy of $\frac{1}{2}$ over the sample.
- Even though we do not predict the exact molecule, the important thing is to have the exact molecule be one of the most probable outputs, we want to give it the best rank possible.

3 Method and approaches

Text Encoders

After scrutinizing the dataset, we hypothesized that leveraging a pre-trained TextEncoder on analogous data could yield superior results. However, in the initial stages, we opted to adhere to the Baseline model's solution, employing DistilBERT [8]. This model, an offshoot of the renowned BERT [3], boasts fewer parameters, enabling us to increase the 'batch_size' and pair

it with an expansive Graph Encoder. Despite this, we explored Hugging Face's transformers[1] and identified a well-received fine-tuned version on the SST-2 dataset [10], which proved to deliver slightly enhanced overall performance.

Once we started to hit a ceiling in terms of performance, we started looking at other options for the text encoding side of the model. As we became familiar with the state of the art on this specific task, **Text2Mol** [4], we learned that both state-of-the-art models, as in the initial model presented in [4] and AMAN [13], used SciBERT [2] as their TextEncoder. This model, based once again on BERT and pretrained on scientific content, looked to be the ideal Text Encoder for our specific task.

The major drawback of using SciBERT was that it proved larger than DistilBERT and restricted our degree of flexibility for the Graph Encoder's dimensionality as well as for the choice of 'batch_size'. Furthermore, we did not observe any relative improvement in our losses or scores as shown in appendix A. As a result, there was no positive balancing the drawbacks we encountered with it, and we decided to drop this TextEncoder and remain with the fine-tuned DistilBERT[2].

Finally, the only small variation we brought was to add a final linear layer to the TextEncoder that we would completely train on our own. Aside from allowing us to fine-tune the encoding, this layer offered us the opportunity to choose an embedding size distinct from the actual BERT which is 768. However, in the end, we chose to keep a size of 768 while continuing to use the linear layer in the encoder.

Graph Neural Networks Models

The library PyTorch geometric[3] provides great implementations of Graph Neural Networks architectures. It was really helpful to test all our ideas. To begin, we started with simple models as in the baseline code like Graph Convolutional Network and we added some layers to increase in complexity.

The code baseline suggested Graph Convolutional Network as a Graph Encoder. However, this model lead to insufficient results. We were looking for an architecture well suited for representation learning. It turns out that PyTorch geometric implements state-of-the-art GNN architectures for representation learning which are Graph Attention Networks first introduced in [12] and then further developed in [3] and [6]. The word "Attention" caught our eyes since we work a lot with this concept in Natural Language Processing. The different architectures deriving from Graph Attention Networks with distinct numbers of layers, with or without the addition of normalization layers and linear layers, gave good results on the test set (around $80-82\%$) . We achieved good performance with the representation of just one model, but we were stuck below 83%. This conclusion lead us to combine the representations of several GNN models, in a boosting method spirit.

Soft voting

In classification or regression, boosting is a Machine Learning technique that combines several weak models to get a stronger one. For instance, if we have a family $(f_{\theta_k}(x))_{1\leqslant k\leqslant K}$ of

[1] `https://huggingface.co/`

[2] `https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english`

[3] `https://pytorch-geometric.readthedocs.io/en/latest/index.html`

models, we can average them to get the model $\frac{1}{K}\sum_{k=1}^{K} f_{\theta_k}(x)$. Averaging allows to reduce the variance and the bias of a classifier or a regressor.

We were inspired by this technique to improve the performance and robustness of our model. Our final model combines the representations of four GNN and Text Encoder models trained independently. For the Text Encoder, we stuck to the *DistilBERT base uncased finetuned SST-2* model and for the GNN models we chose two modified GAT and two Transformer architectures which are further described in appendix B. Ultimately, we calculate the sigmoid kernel or cosine similarity between the embeddings of each graph and text for the four models, followed by averaging these similarities. This approach shares similarities with softvoting, akin to the method of averaging probability distributions from multiple soft classifiers.

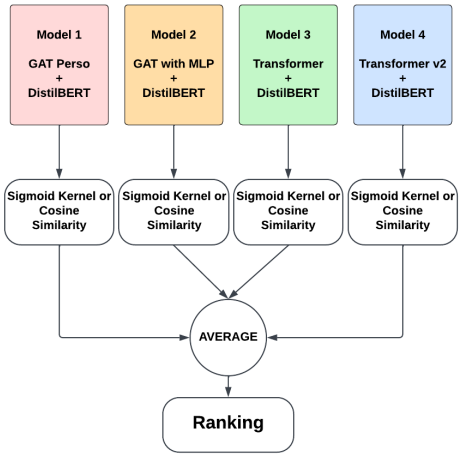


Figure 3: Pipeline of our final model. We compute four embeddings for each graph and text. Then, the pairwise sigmoid kernel or cosine similarities are computed. Finally, the similarities are averaged to get a more robust model.

Training parameters and paradigms

The very first parameter we tried to tune was the 'batch_size'. Indeed, after being limited by our own machines we used specialized GPU in order to increase the 'batch_size'. By trying out both lower and higher sizes than the baseline model - 32 -, we observed that increasing the 'batch_size' improved the overall performance of our models. In this progression, we were unable to identify a maximum point, as our computational resources imposed limitations before any constraints in the progress became apparent. Consequently, we decided to execute our models with a 'batch_size' of 128, striking a balance between reliability (preventing training issues) and optimal performance.

One of the major difficulties we encountered was dealing with over-fitting and a quite high difference between validation scores and test scores. Indeed, we were able to limit the training loss to around 0.05 while our validation remained stuck around 0.40. In the same time, we would reach validation scores higher than 0.90 while our test scores were still around 0.80. Consequently, we made some adjustments and improvements. By testing several values, we learned that for a 'batch_size' of 128, a learning rate of 0.0001 seemed adequate but we added a scheduler to better control the learning throughout the training. For the optimizer, we chose to retain AdamW [7] which was used by the Baseline model. Moving further, we used cross-validation [1], which allowed us to lower the validation loss to the values we were obtaining for the training loss. Specifically, we used 5 folds for the cross-validation.

References

[1] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4, 2010.

[2] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. 2019.

[3] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *CoRR*, abs/2105.14491, 2021.

[4] Carl Edwards, Cheng Xiang Zhai, and Heng Ji. Text2mol: Cross-modal molecule retrieval with natural language queries. 2021.

[5] Raia Hadsell, Sumit Chopra, and Yann LeCun. Contrastive loss. *Cvpr*, 2, 2006.

[6] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. 2021.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Triplet loss. *CVPR*, 07-12-June, 2015.

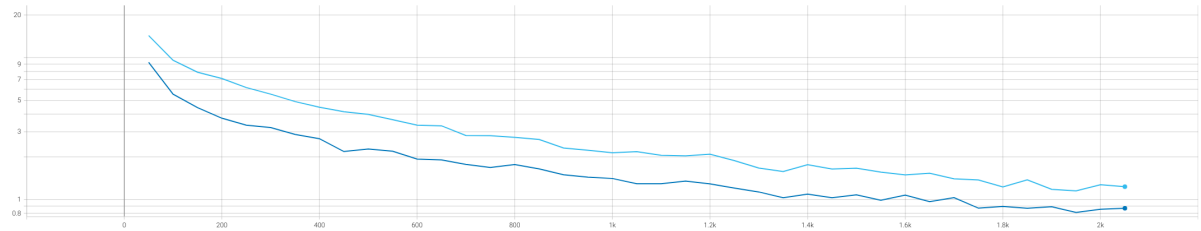
[10] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. 2013.

[11] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. 2016.

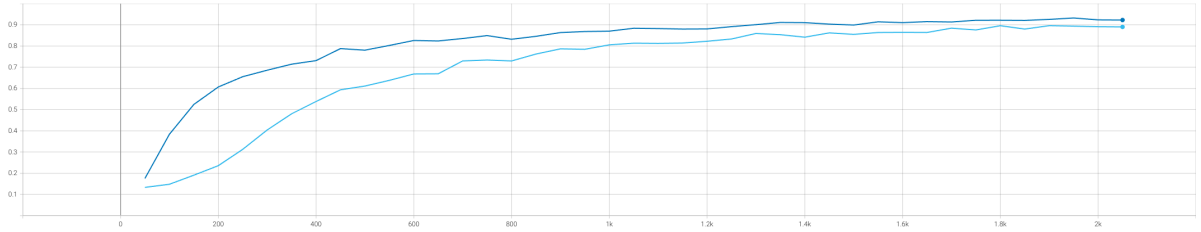
[12] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. 2018.

[13] Wenyu Zhao, Dong Zhou, Buqing Cao, Kai Zhang, and Jinjun Chen. Adversarial modality alignment network for cross-modal molecule retrieval. *IEEE Transactions on Artificial Intelligence*, 2023.

A Choosing the Text Encoder



(a) Training Contrastive Loss



(b) Training LRAP Score

Figure 4: Comparison of Losses and Scores between SciBERT (light blue) and fine-tuned DistilBERT (dark blue) running with the same Graph Encoder.

B Detailed architecture and its performance

B.1 Architecture

The following 4 models were used alongside the DistilBERT Text Encoder on which we added a final linear layer. They were trained on their own for 100 epochs with 5 folds for the cross validations, meaning 20 epochs per fold at a batch_size of 128.

Listing 1: Python Code for the models

```
model_name= 'distilbert-base-uncased-finetuned-sst-2-english'
ModelGATPerso(model_name, n_in=300, nout=768, nhid=1024, n_heads=8, dropout=0.6)
ModelGATwMLP(model_name, nout=768, nhid=768, n_heads=4, n_in=300, dropout=0.75)
ModelTransformer(model_name, n_in=300, nout=768, nhid=768, n_heads=4, dropout=0.6)
ModelTransformerv2(model_name, n_in=300, nout=768, nhid=100, n_heads=2, dropout=0.75)
```

Layer	**Input Shape**	**Output Shape**	**#Param**
TextEncoder	[128, 128], [128, 128]	[128, 768]	66,953,472
(bert)DistilBertModel	[128, 128]		66,362,880
(embeddings)Embeddings	[128, 128]	[128, 128, 768]	23,835,648
(word_embeddings)Embedding	[128, 128]	[128, 128, 768]	23,440,896
(position_embeddings)Embedding	[1, 128]	[1, 128, 768]	393,216
(LayerNorm)LayerNorm	[128, 128, 768]	[128, 128, 768]	1,536
(dropout)Dropout	[128, 128, 768]	[128, 128, 768]	–
(transformer)Transformer			42,527,232
(layer)ModuleList	–	–	42,527,232
(linear)Linear	[128, 768]	[128, 768]	590,592

Table 1: Architecture of the Text Encoder, based on DistilBERT [8] which was trained alongside each of the 4 models

Layer	**Input Shape**	**Output Shape**	**#Param**
GAT Perso	[2000, 2000]	[1, 768]	23,566,080
(relu)LeakyReLU	[2000, 8192]	[2000, 8192]	–
(fc1)Linear	[1, 1024]	[1, 1024]	1,049,600
(conv1)GATv2Conv	[2000, 300], [2, 2000]	[2000, 8192]	4,947,968
(conv2)GATv2Conv	[2000, 8192], [2, 2000]	[2000, 1024]	16,781,312
(fc2)Linear	[1, 1024]	[1, 768]	787,200

Table 2: Architecture of GAT Perso

Layer	Input Shape	Output Shape	#Param
GAT with MLP	[2000, 2000]	[1, 768]	11,458,560
(relu)LeakyReLU	[2000, 3072]	[2000, 3072]	–
(outer)MLP	[1, 768]	[1, 768]	1,182,720
(act)ReLU	[1, 768]	[1, 768]	–
(lins)ModuleList	–	–	1,181,184
(0)Linear	[1, 768]	[1, 768]	590,592
(1)Linear	[1, 768]	[1, 768]	590,592
(norms)ModuleList	–	–	1,536
(0)BatchNorm	[1, 768]	[1, 768]	1,536
(conv1)GATv2Conv	[2000, 768], [2, 2000]	[2000, 3072]	4,730,880
(conv2)GATv2Conv	[2000, 3072], [2, 2000]	[2000, 768]	4,721,664
(inner)MLP	[2000, 300]	[2000, 768]	823,296
(act)ReLU	[2000, 768]	[2000, 768]	–
(lins)ModuleList	–	–	821,760
(0)Linear	[2000, 300]	[2000, 768]	231,168
(1)Linear	[2000, 768]	[2000, 768]	590,592
(norms)ModuleList	–	–	1,536
(0)BatchNorm	[2000, 768]	[2000, 768]	1,536

Table 3: Architecture of GAT with MLP

Layer	Input Shape	Output Shape	#Param
Transformer	[2000, 2000]	[1, 768]	6,036,368
(relu)LeakyReLU	[2000, 3056]	[2000, 3056]	–
(conv)TransformerConv	[2000, 300], [2, 2000]	[2000, 3056]	3,688,592
(fc2)Linear	[1, 3056]	[1, 768]	2,347,776

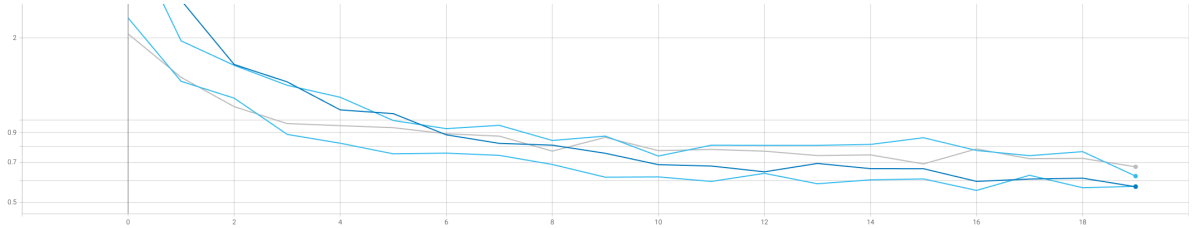
Table 4: Architecture of Transformer

Layer	Input Shape	Output Shape	#Param
Transformer v2	[2000, 2000]	[1, 768]	3,856,068
(relu)LeakyReLU	[2000, 200]	[2000, 200]	–
(res1)ResGatedGraphConv	[2000, 300], [2, 2000]	[2000, 200]	240,800
(Trans1)TransformerConv	[2000, 300], [2, 2000]	[2000, 200]	241,400
(norm1)LayerNorm	[2000, 200]	[2000, 200]	400
(res2)ResGatedGraphConv	[2000, 200], [2, 2000]	[2000, 400]	321,600
(Trans2)TransformerConv	[2000, 200], [2, 2000]	[2000, 400]	322,800
(norm2)LayerNorm	[2000, 400]	[2000, 400]	800
(res3)ResGatedGraphConv	[2000, 400], [2, 2000]	[2000, 800]	1,283,200
(Trans3)TransformerConv	[2000, 400], [2, 2000]	[2000, 800]	1,285,600
(norm3)LayerNorm	[2000, 800]	[2000, 800]	1,600
(MLP)MLP	[1, 800]	[1, 768]	157,868
(act)ReLU	[1, 100]	[1, 100]	–
(lins)ModuleList	–	–	157,668
(0)Linear	[1, 800]	[1, 100]	80,100
(1)Linear	[1, 100]	[1, 768]	77,568
(norms)ModuleList	–	–	200
(0)BatchNorm	[1, 100]	[1, 100]	200

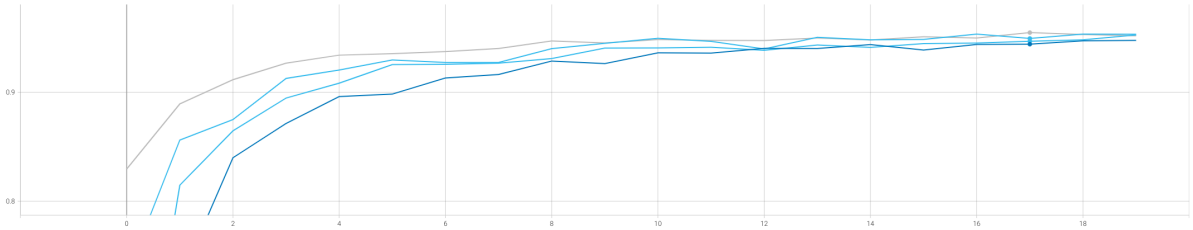
Table 5: Architecture of Transformer v2

B.2 Performance

On all the representations below, we do not provide the association between the colors and the models since the colors are similar and the distinction becomes quite difficult. However, we observe that every model behaves in similar ways in terms of evolution of losses and scores.

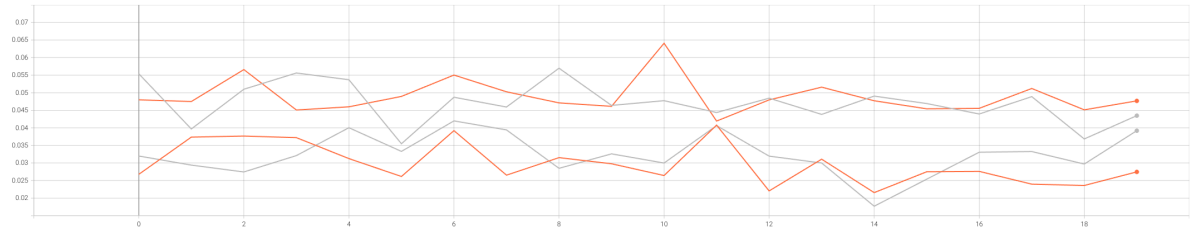


(a) Validation Contrastive Loss

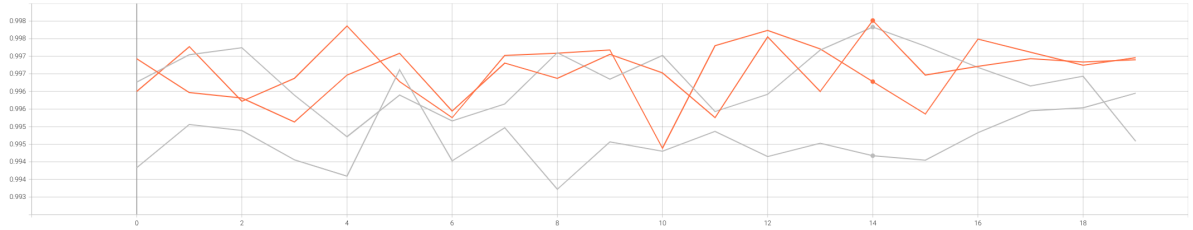


(b) Validation LRAP Score

Figure 5: Validation Losses and Scores of our 4 models during the first fold.



(a) Validation Contrastive Loss



(b) Validation LRAP Score

Figure 6: Validation Losses and Scores of our 4 models during the fifth fold.