ALTeGraD 2023 Data Challenge
Molecule Retrieval with Natural Language Queries

Lilian Hunout, Samy Hocine, Lucas Haubert

École Normale Supérieure, Master MVA
<lilian.hunout, samy.hocine, lucas.haubert>@ens-paris-saclay.fr

February 4th 2024

Natural language query

Cholesteryl linoleate is the (9Z,12Z)-stereoisomer of cholesteryl octadeca-9,12-dienoate. It has a role as a human metabolite and a mouse metabolite. It derives from a linoleic acid.

Structure of the related molecule

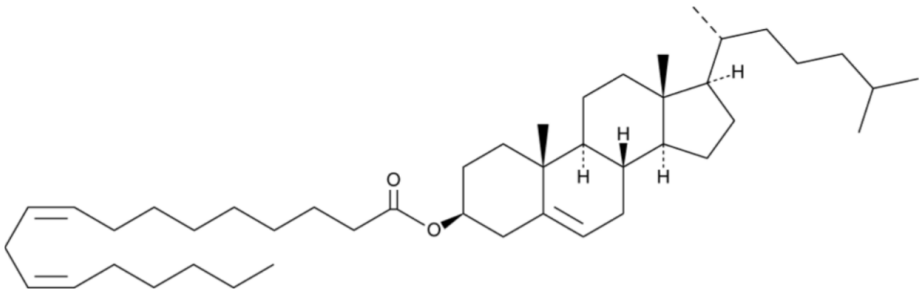

Figure 1: The idea of molecule retrieval with natural language queries

Abstract

This project is relative to the Data Challenge[1] of ALTeGraD course. It explores the text-to-molecule task, aiming to extract molecules based on natural language descriptions. Integrating natural language and molecules poses a complex challenge due to their fundamentally different information encoding. While previous works have addressed text-based and structure-based retrieval separately, this task demands a more direct integration of the two modalities. Moreover, it presents a significant multilingual research challenge, given the unique grammar of molecules. In various scientific domains like medicine or biology, exploiting relationships between textual data and structured ones, such as graphs, can be valuable. By leveraging a matched dataset of molecules and textual descriptions, we train a common semantic embedding space aligned for efficient retrieval. This relevant approach, based on the design of text and graph encoders, offers label ranking average precision (LRAP)[2] performance scores up to 85.68%.

[1]Data Challenge on Kaggle: https://www.kaggle.com/competitions/altegrad-2023-data-challenge
[2]LRAP: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.label_ranking_average_precision_score.html

1 Task Definition

Given a natural language query and a list of molecules, represented by graphs, the goal is to retrieve the molecule that corresponds to the description in the query. With a textual description of a molecule as entry, the model must incorporate the information contained in the description into a semantic representation that can be used to directly retrieve the molecule. This requires the integration of two very different types of information : the structured knowledge represented by text and the chemical properties present in molecular graphs. Figure 1 shows an example of this task for cholesteryl linoleate.

As the two types of data are different, the challenge consists in settling a comparison framework, which is done by means of text and graph encoders. Contrastive learning (CL) is the key feature of such a process: map the similar text-graph pairs close together in the representation space and push other ones apart. Since the dataset[3] of the challenge provide pairs of molecule ID's and textual descriptions, it is assumed that there is only one correct molecule for each description.

Then the Label Ranking Average Precision score (LRAP) is adopted as an evaluation metric of the retrievals. LRAP is relative to the average, for each sample, of the ratio of true labels compared to the total of labels with a lower score. In this context, where there is only one molecule to recover, LRAP is equivalent to the Mean Reciprocal Rank (MRR[4]).

2 Methodology

The methodology is as follows: we will train a text encoder and a graph encoder jointly using contrastive learning (CL), in order to map similar pairs together in a joint representative space. We monitor the progress of training with CL loss and LRAP on the validation dataset at each epoch. Once the training is done, we encode the graphs and text of the test set, calculate the n-to-n cosine similarity matrix and save it in a CSV file we can submit.

2.1 Baseline model

A source code, serving as a reference, is given as an entry point into our project and obtains a score of around 35% in the Kaggle ranking on the public test set. The objective of this challenge will be to maximize this score by modifying the given baseline. In the baseline, `DistilBERT` and `GCN` are used as text and graph encoders. The provided source code was trained with 5 epochs, a batch size of 32 and a learning rate equal to 2e-5. The design of the pipeline, as well as the learning hyperparameters and the overall approach, are subject to change in the following.

2.2 Text Encoder

In our investigation of text encoding strategies for scientific documents, we initially considered leveraging the `DistilBERT` transformer model from HuggingFace as a baseline. `DistilBERT` [10] achieves its efficiency and lightweight nature through a process known as knowledge distillation. This technique involves training a smaller model (the "student") to replicate the behavior of a larger and more complex model (the "teacher"). In the case of `DistilBERT`, the teacher model is `BERT` [3], a highly robust and expressive model for NLP tasks. To further refine our approach towards efficiently processing scientific literature, we incorporated `SciBERT` [1] into our text encoding framework. `SciBERT`, built on the `BERT` architecture, is specifically pre-trained on a broad compilation of scientific texts across diverse disciplines. This pre-training is designed to enhance the model's performance in NLP tasks within scientific contexts, making it ostensibly well-suited for our dataset.

[3]Text-Graph Dataset: https://1drv.ms/u/s!AhcBGHWGY2mukdgpQZ1L-csK3qJ-1w?e=clJXho
[4]MRR Definition: https://en.wikipedia.org/wiki/Mean_reciprocal_rank

In addition to `SciBERT` and `DistilBERT`, we evaluated the performance of HuggingFace's Universal AnglE model [7] over several epochs. As anticipated, `SciBERT` demonstrated a quicker convergence rate, attributed to its training on scientific content similar to our dataset. Conversely, `DistilBERT` and `AnglE` exhibited a slower rate of convergence compared to `SciBERT`. However, `DistilBERT` and `SciBERT` models both ultimately achieved comparable scores/losses upon convergence. Given the lighter architecture of `DistilBERT`, which facilitates training with larger batch sizes and reduces overall model complexity, we favored `DistilBERT` for our text encoding needs. This decision was underpinned by the balance between efficiency and performance, where `DistilBERT`'s streamlined design offered a practical advantage without significant compromise on model effectiveness.

2.3 Graph Encoder

The three graph encoders `GCNEncoder`, `GATEncoder`, and `GINEncoder` share a common goal of encoding graph information into meaningful numerical representations, but they differ in their specific architectures and processing mechanisms.

Commonly, they are all designed to operate on graph data, where nodes represent entities and edges reflect relationships between these entities. Each encoder takes as input graph data such as node features and connections between them, and outputs a dense vector representation of this data, often used in supervised or unsupervised learning tasks.

The similarities between the encoders lie in their general structure. They all feature convolutional layers that perform operations on node features while taking into account the graph's structure. Additionally, they incorporate regularization mechanisms such as Batch Normalization and Dropout to improve generalization and prevent overfitting.

Now, the differences between them mainly lie in the types of convolutional layers used and how these layers operate on graph data:

1. `GCNEncoder` uses `GCNConv` (Graph Convolutional Network Convolution [5]) layers that convolve node features while considering local relationships in the graph. These layers are designed to capture neighborhood information in the graph.

2. `GATEncoder` uses `GATv2Conv` (Graph Attention Network Convolution [13, 2]) layers that incorporate attention mechanisms to weigh the importance of neighboring nodes during convolution. This allows the encoder to focus on specific parts of the graph when extracting features.

3. `GINEncoder` uses `GINConv` (Graph Isomorphism Network Convolution [15]) layers that apply linear transformations to node features, followed by non-linear aggregation functions. Unlike `GCNConv` and `GATv2Conv`, `GINConv` layers do not presume the graph's structure and can thus be more flexible in their representation.

In their operation, all encoders pass node features through multiple convolutional layers, applying activation functions and regularization mechanisms between each layer. They then aggregate node features to obtain a global representation of the graph, often using pooling operations like global mean pooling. Finally, this aggregated representation is transformed into a final output using multi-layer perceptrons and normalization layers to achieve a final, reduced-dimensional representation.

2.4 Loss

For our contrastive learning task between graph representations and their textual descriptions, we adopted a loss based on the Cross Entropy (CE) loss function. This approach aims to minimize the distance between embeddings of similar (matching) items while maximizing the distance for

dissimilar (non-matching) pairs. The essence of employing CE in this context is to ensure that the model accurately distinguishes between correct and incorrect graph-text pairings by assessing the model's predictions against the actual distribution of matches.

For each graph embedding z_{graph_i}, the softmax function is applied across the set of similarity scores $\{s(z_{\text{graph}_i}, z_{\text{text}_j})\}$ for all textual descriptions indexed by j within the batch. Vice versa for each text embedding z_{text_i}. The contrastive loss using Cross Entropy is then defined as:

$$\mathcal{L}_{CL} = -\frac{1}{N}\sum_{i=1}^{N}\left[\log\left(\frac{\exp(s(z_{\text{graph}_i}, z_{\text{text}_i}))}{\sum_{j=1}^{N}\exp(s(z_{\text{graph}_i}, z_{\text{text}_j}))}\right) + \log\left(\frac{\exp(s(z_{\text{text}_i}, z_{\text{graph}_i}))}{\sum_{j=1}^{N}\exp(s(z_{\text{text}_i}, z_{\text{graph}_j}))}\right)\right]$$

This loss function aims to maximize the probability of each graph embedding matching its correct textual description (and vice versa) relative to all other pairs in the batch, thereby encouraging the model to learn discriminative embeddings that accurately reflect the correspondence between graphs and textual descriptions.

2.5 Optimizations

We encountered several technical limitations, including training on a single *Tesla P100 PCIe 16 GB* and a time restriction of maximum 30 hours per week. We quickly realized that this time was valuable to train our models as much as possible. To overcome these challenges, we implemented several optimizations in the code to improve the performance and efficiency of the training process. First, we optimized the DataLoader using the Torch library, which loaded data more efficiently and reduced wait times during training. Additionally, we have incorporated the use of autocast mixed-precision, a powerful technique for exploiting the benefits of 16-bit floating point calculations, providing significant speedup of calculations without compromising the quality of results. Additionally, we adopted a Hugging Face accelerator with the Hugging Face library, which made it possible to fully exploit modern hardware capabilities to accelerate the training process.

These optimizations were crucial, especially considering that without them we were experiencing CUDA memory errors when processing large volumes of data. With these adjustments, we were able to increase batch sizes, optimize the code, and achieve faster training times, contributing to better overall model efficiency.

2.6 Ensemble Approach

After studying the basic models, we found that the correct molecule was very frequently found among the top-ranked molecules. A part of the matches was found jointly by all the base models. On the other hand, some matches were only found by certain models. This is because graph encoders do not extract features in the same way. Our objective was to take advantage of all the capabilities of the different models using an ensemble approach. This approach consisted of averaging the results obtained for each model (n-to-n cosine similarity matrix). We noticed that normalizing or not the matrices on their rows did not have much impact on the result of the operation.

3 Experiments

3.1 Data and Evaluation

The framework of the Data Challenge provides a host environment on Kaggle, as well as text-graph data. In particular, files *train.tsv* and *val.tsv* are training and validation pairs of graph ID's and textual descriptions. The tokenizer construction involves file *token_ embedding_ dict.npy* which is a dictionary that maps tokens from molecule substructures to their embeddings. Also,

graphs are stored in a folder containing *cid.graph* files that are made of edge lists and node to token mappings. Finally, test files, *test_ text.txt* and *test_ cids.txt*, contain textual descriptions and ID's to be mapped, the goal being to build a get a correspondence between each others.

These datasets are therefore already separated into train(80%)/validation(10%)/test(10%) distributions. We kept this distribution by making sure to shuffle the data during training. Alignment models are trained on the training data and the results are evaluated by searching all molecules in the dataset.

3.2 Results

To train the models, we use Adam optimizer [4] and two different learning rates. The `DistilBERT` and `SciBERT` models uses a finetuning learning rate of 3e-5, as used in [3]. The rest of the model uses 1e-4 as used in [12]. We use a StepLR scheduler with a reduction in the learning rate every 5 epochs by 50%. We train for 40 epochs with a batch size of 64. We use the first 256 text tokens for the text encoder. It was possible to upgrade to a batch size of 128 using `DistilBERT`.

3.2.1 Baseline Models

Encoder models based on `GCN`, `GAT`, `Stacked-GAT`[5] and `GIN` layers both show similar performance. The results on the training and validation datasets are presented in Table 1. Encoders based on `Stacked-GAT` and `GIN` layers with more weight performed slightly better. We believe that the similarity in performance between these models is because description is a bottleneck.

We notice better LRAPs in validation than in training. This is linked to the fact that the training dataset is 8 times larger and it is therefore difficult to maintain a high rank in the middle of an increasing number of data, leading to the decrease in the score. We can therefore conclude that the value of the LRAP score will be very dependent on the size of the dataset on which it will be calculated.

We tried to freeze the first attention modules in `DistilBERT` model. Although faster, it did not lead to good results.

3.2.2 Ensemble

We find that the ensemble method shows significant performance improvements. The model ensemble increases the validation LRAP by approximately 9.5% compared to the average of the baseline model scores. It should be noted that the hyperparameters of these models are exactly the same and that the models learn different ways of classification which are complementary. The ensemble approach may incorporate different encoder architectures and recovery schemes, which may have different understandings of how to solve the problem. We find that the combination of several architectures is more efficient.

Table 1: Results

Model	Training LRAP	Validation LRAP
`DistilBERT + GCN`	54.26%	75.10%
`DistilBERT + GAT`	54.53%	74.67%
`DistilBERT + Stacked-GAT`	57,68%	77.67%
`DistilBERT + GIN`	57.49%	77.02%
All-Ensemble	NaN	85.68%

[5]GATEncoder where the GATv2Conv layers all have 2 multi-head-attentions

4 Conclusion and Future Work

In this work, a study of the relevance of various text encoders have been made, leading to adopt `DistilBERT` as the text encoder, due to its practical advantages. An ensemble method have been implemented in order to combine various models such that `GCN`, `GAT` and `GIN`, leading to significant results on LRAP scores, up to 85,68%.

Looking forward, there are several avenues we are excited to explore to refine our model's performance further. Future directions could involve delving deeper into data augmentation techniques (cf. Appendix B). By augmenting our batch sizes, we anticipate a more efficient training process, allowing our models to learn from a larger volume of data simultaneously. Indeed, in most of the previous works on a similar task such as [11], the batch size is much higher than ours. Extending the training duration presents an opportunity to deepen the model's learning, potentially uncovering more nuanced patterns within the data. Additionally, adopting a cosine scheduler for learning rate adjustment could introduce a dynamic element to our training regimen, helping the model escape local minima and converge towards the global minimum.

Another strategy to enhance our model's generalizability involves reshuffling and re-splitting the combined training and validation datasets. This process aims to minimize the risk of potential bias and ensure a more representative distribution of data between the training and evaluation phases.

The intersection of textual and molecular data through machine learning not only opens new avenues for scientific exploration but also underscores the continuous nature of learning and adaptation in the field of AI.

References

[1] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *EMNLP*. Association for Computational Linguistics, 2019.

[2] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[6] Romain Lacombe, Andrew Gaut, Jeff He, David Lüdeke, and Kateryna Pistunova. Extracting molecular properties from natural language with multimodal contrastive learning, 2023.

[7] Xianming Li and Jing Li. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*, 2023.

[8] Yangguang Li, Feng Liang, Lichen Zhao, Yufeng Cui, Wanli Ouyang, Jing Shao, Fengwei Yu, and Junjie Yan. Supervision exists everywhere: A data efficient contrastive language-image pre-training paradigm, 2022.

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[10] 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.

[11] Bing Su, Dazhao Du, Zhao Yang, Yujie Zhou, Jiangmeng Li, Anyi Rao, Hao Sun, Zhiwu Lu, and Ji-Rong Wen. A molecular multimodal foundation model associating molecule graphs with natural language, 2022.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

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

[14] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks, 2019.

[15] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[16] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations, 2021.

A Graph Convolutional Layers

A.1 Graph Convolutional Network (GCN)

In a Graph Convolutional Network (GCN) [5], the convolutional layer is a message passing layer. Its formula is given by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where:

- $H^{(l)}$ denotes the node features at layer l, with $H^{(0)} = X$ being the input features.
- $\tilde{A} = A + I_N$ is the adjacency matrix of graph G augmented with self-connections, A being the original adjacency matrix and I_N the identity matrix of size $N \times N$, where N is the number of nodes.
- $\tilde{D}$ is the degree matrix of $\tilde{A}$, with diagonal elements $D_{ii} = \sum_j \tilde{A}_{ij}$.
- $W^{(l)}$ is the weight matrix for layer l, learned during training.
- σ represents a nonlinear activation function, such as ReLU.

The Graph Convolutional Network (GCN) leverages the spectral theory to efficiently capture the topology of a graph. It aggregates the features from the neighbors of a node, thereby embedding the local graph structure into the node representation.

A.2 Graph Attention Network v2 (GATv2)

In Graph Attention Networks (GATs) [13], the key feature is the attention mechanism that computes the hidden representations of each node by attending over its neighbors. GATv2 [2] addresses the limitations of the first version by enabling the attention mechanism to directly interact with the transformed features. This change significantly increases the expressiveness of the attention coefficients, allowing them to adapt more effectively to the features' transformed space. Its formula is given by :

$$\alpha_{ij} = \text{softmax}_j\left(\text{LeakyReLU}\left(\mathbf{a}^T[Wh_i^{(l)} \| Wh_j^{(l)}]\right)\right)$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \cdot Wh_j^{(l)}\right)$$

where:

- $h_i^{(l)}$ is the feature vector of node i at layer l.
- $\mathcal{N}(i)$ denotes the set of neighbors of node i.
- α_{ij} are the computed attention coefficients, which depend on the transformed features of the nodes, making the attention mechanism more expressive.
- W is the weight matrix applied to node features before the attention coefficients are computed, a key difference from `GAT` where the attention is computed prior to this transformation.
- σ represents a nonlinear activation function.

A.3 Graph Isomorphism Networks (GIN)

Graph Isomorphism Networks (GINs) [15] are designed to capture the power of the Weisfeiler-Lehman (WL) graph isomorphism test. The update rule for a node's representation in GINs is given by:

$$h_i^{(l+1)} = \text{MLP}^{(l)}\left((1+\epsilon^{(l)}) \cdot h_i^{(l)} + \sum_{j \in \mathcal{N}(i)} h_j^{(l)}\right)$$

where:

- $h_i^{(l)}$ represents the feature vector of node i at layer l.
- $\mathcal{N}(i)$ denotes the set of neighbors of node i.
- $\epsilon^{(l)}$ is a learnable parameter or fixed scalar used to weigh the importance of a node's own features relative to its neighbors'.
- $\text{MLP}^{(l)}$ represents a multi-layer perceptron applied at layer l.

B Graph/Text Augmentation

To improve the performance of our model, we would need to increase the data. As explained in [11], given the small amount of data we have for graph-text contrastive learning compared to the image-text contrastive learning presented in CLIP [9], increasing the data could greatly improve the training of our model. In fact, [8] proposes to use data augmentation to improve the training of the CLIP model. They achieve similar performance to the original CLIP model with 7 times less data. To apply this method to contrastive learning graph-text, MoMu [11] proposes to use the graph augmentation techniques of [16]. The different graph augmentation methods are more or less relevant depending on the data represented by the graphs. For graphs of biological molecules, the most effective methods according to [16] are node dropping and subgraph generation. These are also the methods suggested in [11] and [6] to achieve text-graph contrastive learning.

For text data augmentation, the global approach suggested in [8] and used in the other articles to augmenting text utilizes the EDA technique proposed in [14], which includes a trio of augmentation processes: substituting synonyms, shuffling words randomly, and eliminating words at random. For each instance of text, one out of these three processes is selected at random to apply the augmentation.

In concrete terms, for each pair ($\mathcal{T}_i$, $\mathcal{G}_i$) of each batch of size N, we create 2 noisy text descriptions using EDA ($\mathcal{T}_i^1, \mathcal{T}_i^2$) and two graphs ($\mathcal{G}_i^1, \mathcal{G}_i^2$) corresponding to the noisy molecule using node dropping and subgraph. Then, we compute their embedding through text and graph encoder ($z_{\mathcal{G}_i}^1, z_{\mathcal{G}_i}^2, z_{\mathcal{T}_i}^1, z_{\mathcal{T}_i}^2$). Finally, the loss function is calculated by averaging the losses for each of the pairs ($z_{\mathcal{T}_i}^p, z_{\mathcal{G}_i}^j$) for $(p, j) \in \{1, 2\}^2$.

At the end of the project, we tried to use this method of data augmentation. However, we had difficulties with the implementation due to the higher cost in GPU memory, and we were unable to succeed. This is therefore an area for improvement.