ALTEGRAD Challenge 2024 on Molecules and NLP: contrastive learning applied to retrieval systems of chemical compounds

Balthazar Neveu
ENS Paris-Saclay
Saclay, France
balthazar.neveu@ens-paris-saclay.fr

Léa Khalil
Ecole Polytechnique
Palaiseau, France
leakhalil@yahoo.fr

Basile Terver
Ecole Polytechnique
Palaiseau, France
terverbasile@gmail.com

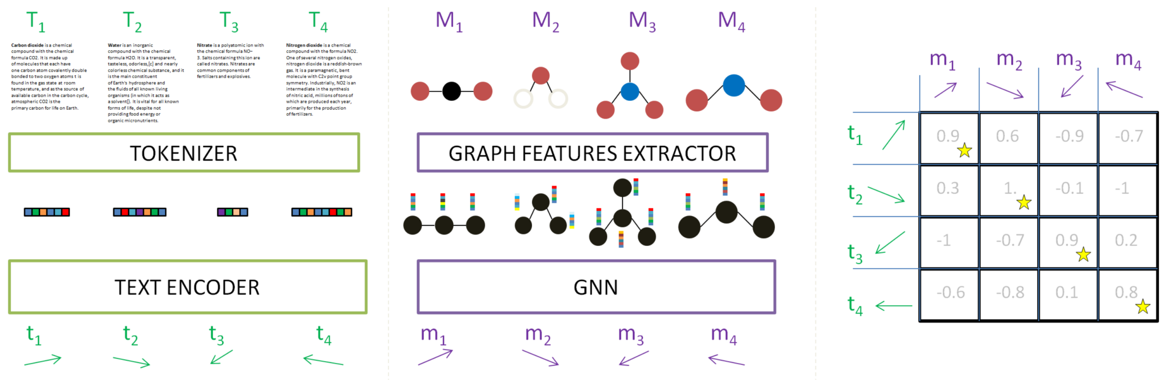


Figure 1: Our system matches a text description with the most relevant molecules. On the left side, text descriptions T_i are tokenized and transformed into a numerical representations (embeddings) t_i using a large language model. In the center, molecules M_j are represented using graphs where each node is a molecular structure (not just an atom) attached with its vector representation. Graphs are processed by a graph neural network to create embeddings m_j. On the right side, pairwise similarity matrix between text and molecule allows finding the best matches (ranking) and is also the central mathematical object used to train the system (contrastive learning means maximizing $\vec{t_i}.\vec{m_i}$ and minimizing $\vec{t_i}.\vec{m_j}\ \forall j \neq i$).

KEYWORDS

contrastive learning, graph neural networks, large language models

This study aims at building meaningful vector representations of text descriptions and molecules. The challenge is to consistently embed these two modalities into the same vector space. The most direct application is a retrieval system for chemical compounds based on text descriptions. Good molecule representations are more general as they may be re-used in other downstream tasks (interesting chemical properties may emerge from the molecule embeddings we get). We used a pretrained Large Language Model (LLM) and a Graph Convolutional Network (GCN) to map the two modalities into a shared embedding space and trained our models using contrastive learning. Our best model achieves a LRAP score of **0.905** on the provided challenge test set.

1 INTRODUCTION

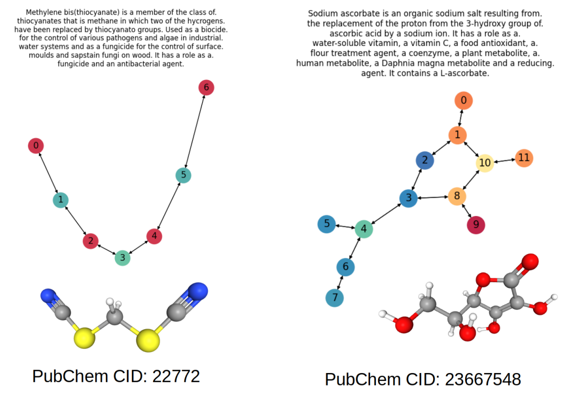
Recent advances in contrastive learning have allowed creating powerful representations for various modalities which can be embedded in the same shared vector space. CLIP [10] learns image representations which match with text description. *Can the same idea be extended to molecules and text?* Text-2-Mol [6] introduced the groundings of the problem of pairing a text description with the corresponding molecule. We use the dataset which came along with this paper and our problem is restricted to using this data solely.

Solving this problem requires having a model with a good understanding of natural language. We leverage the language capabilities inherent to pretrained Large Language Models (LLM) to create embeddings for text descriptions. We work with models that have good understanding of the English language. The difficulty stands in how we guide the language model to detect text features (understanding of the chemical domain) which will match the molecule patterns. To achieve this, a GNN (graph neural network) is trained along the LLM to extract information from the molecule graph structure which should be as close as possible to the text representation.

2 CONTEXT

2.1 Dataset

The provided dataset is made of pairs of molecules and text descriptions. Text-2-Mol [6] came out with a dataset of 33 310 molecule paired with a textual description, made publicly available (which is always appreciated among the research community). Molecular structures (unitary structures slightly more generic than the atom level) have been isolated with their corresponding vector representations. Vector representation relies on Mol2Vec[7] which is a machine learning algorithm to create vector representations of molecular substructures trained in an unsupervised manner over 19.9 millions of chemical compounds. In figure 1, this step corresponds to the graph features extractor step. Each node is paired


Methylene bis(thiocyanate) is a member of the class of. thiocyanates that is methane in which two of the hycrogens. have been replaced by thiocyanato groups. Used as a biocide. for the control of various pathogens and algae in industrial. water systems and as a fungicide for the control of surface. moulds and sapstain fungi on wood. It has a role as a. fungicide and an antibacterial agent.

Sodium ascorbate is an organic sodium salt resulting from. the replacement of the proton from the 3-hydroxy group of. ascorbic acid by a sodium ion. It has a role as a. water-soluble vitamin, a vitamin C, a food antioxidant, a. flour treatment agent, a coenzyme, a plant metabolite, a. human metabolite, a Daphnia magna metabolite and a reducing. agent. It contains a L-ascorbate.

Figure 2: 2 samples of molecules compounds paired with their description (matching correctly with their unique identifier on PubChem). These two examples show the difficulty of the task, both descriptions include the word Water being used in very different contexts. The Mol-2-Vec token ids (represented with colors in the graph) model more than atoms (molecule structures in their context neighborhood)

Split	Train	Validation	Test
Split ratio	80%	10 %	10%
Graph/Text pairs	26408	3301	3301
Groundtruth	Yes	Yes	Not public

Table 1: Dataset split

with a vector of length 300 which naturally contains a lot of information about the nature of the molecular substructure (this feature representation is far more powerful than simply depicting a single atom for instance). In the Mol-2-Vec dictionary, there are 3137 tokens types (+1 unknown), each token id corresponds to a vector of dimension 300. The provided dataset has been split into 80% , 10%, 10% (train, validation, test - see table 1).

2.2 State of the art

In the following section, we'll briefly describe relevant work related to our topic (a more thorough study is available in Appendix C). CLIP [10] builds image representation embedded in the same vector space as textual representations and works with augmented pairs of image/text by batches of size 32000.

Text-2-Mol [6] applies the same concept to molecules represented as graphs (rather than regular image grids) and uses graph convolutional networks (GCN [9]) to extract meaningful information. Text-2-Mol uses tiny batches of size 32 compared to CLIP. Text-2-Mol fine tunes SciBERT (a transformer encoder [1]) pretrained by masked language modeling on a corpus of 1 million of scientific articles.

Molecule augmentations (atom masking, bounds deletion, subgraphs deletion) are proposed in Mol-CLR[13], a contrastive learning approach applied to molecule graphs only (like SimCLR [3] or the recent DINO [2] which work solely on images, but here transposed to molecules).

3 FRAMEWORK

This project involves processing a large amount of data, requires a lot of computational power and working as a team with heterogeneous environments. Since we had anticipated the need to make a lot of experiments, we carefully designed a framework which allows training and to keep a record of all experiments while having a reproducible code. All explanations on the framework we developed have been listed in appendix B . As a matter of fact, getting an accurate system for this task was also a software engineering challenge and not only a pure machine learning task. Our code is available on GitHub for future work.

4 OUR WORK

Base model

Figure 1 depicts our approach to this task: Text descriptions are transformed into sequences of tokens of various lengths (sequences of integers). Tokenized sequences are then embedded into a vector space using a large language model encoder. Throughout the project, we used **various versions of the BERT**[5] architecture (which is the encoder part of the Transformer [12] which have been pretrained with a masked language modeling task). Each description T_i is transformed into a vector t_i (text descriptions embeddings) by extracting the representation of the **first token** (the CLS token).

Molecules M_i will be transformed into vector descriptors m_i (molecule embeddings). Each molecular structure (more generic than an atom) is first transformed into a vector using Mol-2-Vec [7] which incorporates a lot of prior knowledge about chemistry into each node vector. The whole molecule structure is kept as an undirected graph to model the bounds between elementary molecular structures. **A graph convolutional network (GCN)** is then used to embed these graphs into a vector space. The molecule and text descriptions embeddings are then compared.

Contrastive learning is used to train such a model: supervision comes from the knowledge of the pairing between the molecule and its text description. We compute a pairwise similarity between the molecule embedding m_j and text embeddings t_i and maximize it when $i = j$.

Categorical cross entropy for contrastive learning: The loss we're using for the baseline is $L(m,t) = \text{CCE}(m^T.t, \mathbb{I}_b) + \text{CCE}(t.m^T, \mathbb{I}_b)$. where $t, m \in \mathbb{R}^{d,b}$ with embedding dimension $d = 768$ are respective batches of text and molecule embeddings of size $b = 32$.

$\text{CCE}(m^T.t, \mathbb{I}_b) = \sum_{i=1}^{b} l_i$

with $l_i = -\ln(\text{softmax}(m^T.t_i)) = -\ln \frac{e^{m_i^T.t_i}}{\sum_{j=1}^{b} e^{m_j^T.t_i}}$

Preliminary study (baselines)

We begin with a few toy experiments to see which architecture factors are most promising (initial hope is that the performances will scale accordingly when we add all extra machine learning tricks).

Experiment ID	Model Size	LLM	GNN	LRAP
101	593k	Frozen Distill-Bert	Base 3 layer GCN	18.7%
106	964k	Frozen Distill-Bert + Adapter	Base 3 layer GCN	26.8%
114	2.125M	Frozen Distill-Bert + Adapter	Big 5 layer GCN	31.6%
112	964k	Frozen Sci-Bert + Adapter	Base 3 layer GCN	36.7%
113	2.125M	Frozen Sci-Bert + Adapter	Big 5 layer GCN	39.8%
65	66.9M	Trainable Bert	Base 3 layer GCN	63.5%
400	110M	Trainable Sci-Bert	Base 3 layer GCN	66%

Table 2: Base models specifications and performances - Trained with batches of size 32 using the Adam optimizer [8] (Learning Rate $= 10^{-4}$ **Weight decay** 10^{-2}**) to minimize the Categorical Cross Entropy loss applied to contrastive learning.**

Frozen LLM weights: We started with simple models based on the base GCN (3 graph convolution layers followed by a global pooling layer and 2 layers MLP). Instead of fine tuning all parameters of the LLM, we first froze the LLM parameters. Although simple, this idea intuitively has many advantages for training:

- We discard the huge memory cost of training a LLM (memory issues not only come from storing the weights on the GPU but from all the optimizers variables during back propagation). *The idea could have been pushed further by precomputing the text embeddings and storing them on disk.*
- Intuitively, freezing the LLM parameters should make the training more stable as the LLM embeddings acts as a kind of anchor that the GNN shall match.

Unfortunately, training achieves a low LRAP score (*not too surprisingly since the number of parameters to train is relatively low*). We added an "adapter" module which is simply a MLP which project the text representations into a more adapted space which can be matched with the graph embeddings.

Influence of the graph neural network size: We pursued our explorations to see the impact of the GNN size. The **big GCN** (5 graph convolution layers with 2 residual connection) is more complex and has more parameters to train. We can see that the LRAP is improved by increasing the GCN size.

- Using frozen Distil-BERT: from experiment 106 (base GCN LRAP = 26.8%) to 114 - (big GCN LRAP = 31.6%)
- Using frozen SciBERT: from experiment 102 (base GCN LRAP = 36.7%) to 113 - (big GCN LRAP = 39.8%)

Influence of the pretrained language model: We also browsed Hugging Face to find models that could be dedicated to scientific-specific language processing. Sci-Bert[1] (used in Text-2-Mol[6]) can be used as a drop-in replacement for the Distil-Bert baseline model. Improvements were two-fold when changing the pretrained LLM during this preliminary study: a tokenizer dedicated to a scientific corpus (SciVocab) seems by nature a natural choice for scientific words... (*molecule names not being too frequent in common language*). The Sci-Bert model may also have reasoning capabilities closer to science and chemistry reactions as it's been trained on one million scientific papers (a total of over 3 billions of tokens, similar to BERT). This was translated by a improvement in performances. Increasing the GNN size improves accuracy.

- Using the Base GCN : from experiment 106 Distil-BERT (LRAP = 26.8%) to experiment 112 - SciBert (LRAP = 36.7%).
- Using the Big-GCN: from experiment 114 Distil-BERT (LRAP = 31.6%) to experiment 113 - SciBert (LRAP = 39.8%).

The capacity of the network (same number of parameters) being fixed between these two experiments, it proves that the SciBERT pretraining with SciVocab tokenization is definitely more suited for our task. We hoped that this > 8% LRAP improvement would be translated when training a fully trainable LLM.

Unfortunately we later conducted larger experiments with fully trainable LLM (starting from the pretrained weights) and the performances improvements due to the previous modifications did not fully apply. Using the Base GCN : from experiment 65 Distil-BERT (LRAP = 63.5%) to experiment 400 - SciBert (LRAP = 66%), there's not that *seq*8% LRAP improvement that we had seen earlier. Furthermore, it's hard to tell whether the 2.5% improvement is due to the SciBert model pretraining being more suited for our task or the fact that the model has nearly twice as many trainable parameters than the Distil-BERT model.

This preliminary study gave us guidance that using a larger GCN and using SciBERT pretrained weights and tokenizer were good trends to follow to try improving our results. The pitfall is that fine tuning SciBERT comes with a bigger memory footprint than Distil-BERT which requires diminishing batch sizes for a fixed GPU (large batch size seems to be one key factor of the success of contrastive learning when looking at CLIP [10]). Experiments 65 and 400 have been cautiously trained with *the same batch size of 32* and same hyperparameters to get comparable results:

- the SciBERT experiment 400 could only be ran using a NVIDIA RTX A4000 with 24Gb of RAM
- the Distil-BERT experiment 65 was possible on a NVIDIA Tesla T100 with 16Gb of RAM (Kaggle Kernels notebook).

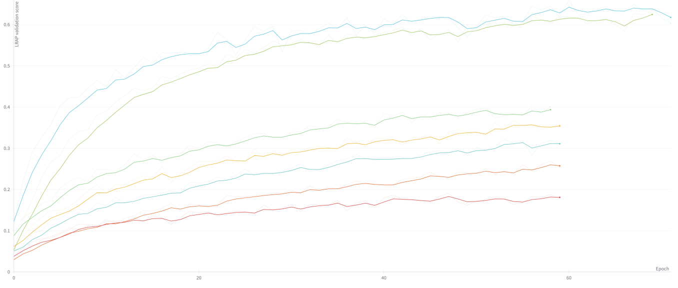


Figure 3: Training curves for the preliminary study.

Our model

We have trained and experimented with several models throughout this project. We identified axes which left room for improvement of the LRAP score and we tried to combine them to create the most robust and efficient model.

Research on the LR parameter.
When running tests on the baseline with different hyperparameters, it became obvious that the model was very sensitive to the learning rate. We empirically identified an ideal learning rate parameter to train our best experiment on the baseline. In experiment 68 we trained the baseline model with this optimal learning rate of $lr = 7e - 5$ and a batch size of 128 and achieved 68% of accuracy.
We also implemented a learning rate scheduler. We tried cosine annealing which did not prove very efficient and ended up working with a plateau learning rate scheduler for most of the following experiments.

LoRa.
Because of the nature of the LRAP metric, a key element to improve the LRAP was to be able to increase the batch size, which was hard to do here given the nature of the data. We implemented LoRa to reduce the number of parameters which need to be trained for the LLM (only 1% left approximately) and to be able to increase the batch size during training.
This technique however did not prove as efficient as expected. It led to no major improvement on the batch size when using Scibert (we were not able to go beyond batches of size 64). When adding Lora to the baseline model and using Scibert (experiment 502) with a batch size of 64 (the biggest we could fit on the GPU) and a learning rate of 3.10^{-4}, we only obtained a 60% LRAP score, which was not an improvement over the best baseline model mentioned in the previous subsection. The same baseline model with DistilBert and batch size 128 (experiment 514) achieved a LRAP score of 68% matching the previous model. This also showed that using SciBert was not necessarily helping achieve better accuracy but rather adding constraints to our trainings regarding batch sizes.
Using LoRa proved more efficient when using DistilBert, where it enabled us to train experiments with bigger batches. Experiments 577 and 9011 have the same architecture (5 layer GCN, learning scheduler with the same parameters, DistilBert) except for the fact that 577 uses LoRa and trains batches of size 192 and 9011 trains simply batches of size 180 without LoRA. In the end, we noticed a gap in performance since 9011 scored a LRAP of 86% while 577 only scored 74%, the bigger batch size did not make up for the performance decrease caused by the reduction of trainable parameters of the LLM.
This led us to use LoRA when experimenting on other axes such as the influence of the GCN architecture or the learning rate scheduler and take advantage of the increased speed of the training, but to drop it for experiments with a high LRAP score potential. However, in some situations which we cannot fully explain we still obtained good results when training with Lora and Scibert with architectures such as that of experiment 573 which achieved 80% in LRAP score and which we reused later. (See Appendix A.1 for detailed information on the experiments)

GCN.
As previously mentioned, we had identified the GCN as a promising work direction to improve the accuracy of our model. After testing the difference with a Frozen LLM, we compared the different GCN architectures when using SciBert and LoRa for the LLM. This confirmed our intuitions, namely that accuracy improves when we increase the size of the GCN for otherwise similar models. The results and architectures of these experiments can be found in Appendix A.2 The gap in performance can however be reduced when increasing the size of the batches during the training for models with a smaller GCN architecture.
After this round of experiment, we stopped working with Scibert and used only DistilBert. We ran several very successful experiments by toying with hyperparameters such as the batch size and the architecture of the GCN. The models, architectures and scores are detailed in Appendix A.3. We noticed that we could make up for the lack of performance of the "bigger GCN" compared to the "Fat GCN" by increasing the batch size. The "Fat GCN" was also slower in training and could not be scaled as easily. The results of these four experiments were all used for our final model and included in the ensembling.

Loss functions for the training.
All previous experiments were trained using the contrastive loss . Following the idea of Text-2-Mol [6], we changed the training loss. We replaced the categorical cross-entropy loss by one derived from the binary cross entropy. In doing so, we consider that our problem is no longer a multi-class problem (one label per pair of molecule and description) but rather a binary classification problem. This loss will be refered to as the 'binary contrastive loss' both in this report and our code. We repeated the baseline experiments with this new loss with different batch sizes in experiments 18, 19 and 20 which are detailed in Appendix A.4.1. We noticed a big improvement over the 66% we had achieved on the baseline when training on the contrastive loss. We also observed that the trainings were even more sensitive to the learning rate parameter and the performance did not scale as well as hoped when increasing the batch size.
We reproduced variants of the experiments 9008 to 9011 which had yielded high scores when trained with the contrastive loss. Again, by playing with the GCN architecture, the batch sizes and adding a learning rate scheduler, we were able to launch some effective experiments. After 200 epochs, our most successful experiments 9075 and 9077 had an LRAP score of over 86% on the valid set, and were trained with a smaller batch size than 9008-9011. Their descriptions and scores are visible in Appendix A.4.2. We included the outputs of 9075 and 9077 in the ensembling which was used to elaborate our final best model.

Freezing the LLM, training a large GCN from scratch.
We have finally experimented with an improved version of the idea we had at the beginning of freezing the weights. We reused models we had previously train jointly which had yielded correct LRAP scores. We assume that the LLM is able to create rich text features which are relevant to our molecule corpus. We froze the weights of the pre-trained model in order to reduce the GPU memory used during training. We are now able to train a larger GCN from scratch (or fine tune our pre-existing one with bigger batch sizes, see table 3).

LLM	Batch Size	GPU Utilization	GPU Memory
Trainable	8	90%	2.95 GB
Frozen	8	91%	0.913 GB
Frozen	32	100%	1.29 GB
Frozen	128	100%	2.97 GB

Table 3: Comparison of memory occupancy for different model configurations on NVIDIA T500 with 4GB of GPU Memory. GCN Model is the FatGCN. LLM is a SciBERT model that we trained from scratch with LORA. This allows putting much larger batch sizes in the GCN model (from batches of size 8 when training LLM and GCN jointly to 128 - please note that the trainable LLM memory in the first row is already reduced a lot by using LoRA compared to traing all parameters at once).

This idea allowed us to start from a language model pretrained using SciBERT (and FatGCN) using batches of size 64 (trained on a Nvidia A5000) which was stuck at a LRAP score of 0.8 (experiment 753). We trained a new GCN from scratch using batches of size 256 leading to 0.87 LRAP score (trained on a much smaller GPU RTX 2090)(see Appendix A.4.2 for more information)

4.0.1 Final Model. In the end, after having extensively experimented with different architectures by changing the GCN, the LLM, the batch size and the learning rate, we obtained seven different models with acceptable LRAP scores: 9008, 9009, 9010, 9011, 9075, 9077 and 611. We applied an ensembling technique to average the scores of each of these models which leads to an extra LRAP score improvement as suggested in [6]. (average submission csv for experiments 9008, 9009, 9010, 9011, 9075, 9077 and 611 basically, a trick which has no extra cost and allows taking more confident decisions). Our final model averages the outputs of seven different models which each take 12 to 13 hours: 91 hours of computation were hence required to obtain this model. The best LRAP score of any single one of these models was around 86.5% and we managed to reach a score above 90% thanks to this technique.

5 DISCUSSION

5.1 Limitations

It was very difficult to find the right trends regarding which hyper parameter, architecture or loss tweak would have an impact on the final results. We mostly tried to follow a rational path toward improvements but it almost feels like a random search in that space would have been nearly as efficient. This is a difficult problem. We first considered designing a toy example for text/graph contrastive learning by synthesizing graphs from colored primitive geometric shapes associated with text descriptions (*(for instance: blue triangles placed regularly at the ends of a red star with seven branche)*). But it was too unsure that getting intuition from a toy example would have scaled to the molecule problem.

5.2 Improvement ideas

While working on this project we had several ideas which we did not have time to explore but that we think could lead to interesting results.

5.2.1 Reducing memory footprint. We have deployed mixed precision training and we did a few unsuccessful attempts with QLORA to reduce the language model RAM occupancy but we still believe that reducing the memory footprint could benefit to training performances (either to increase batch size as suggested by [10], have larger models or even run the training on smaller GPU resources).

One unexplored idea is alternating gradient descent between models (graphs and LLM). As the LLM and the GNN model are trained jointly, all parameters and optimizer parameters need to be kept in GPU memory. We could try to alternate between training only the LLM while freezing the graph model for a few steps (we would not expect the LLM to change much anyway), then move back the parameters to CPU and alternate with training the GNN model. This would potentially allow to increase batch size further (you may have to start from a pretrained baseline to avoid a difficult initialization).

5.2.2 Single-modality pretraining.
(Idea, Not explored): We could have fined tuned the LLM on the text corpus of molecules descriptions only in a masked language modeling task. This could have allowed to have a better understanding of the text corpus and potentially better text embeddings. This could either have been incorporated as a pre-training step or as an auxiliary task to the contrastive learning task (additional loss for masked language learning). In the same gist, we could also pre-train the GCN. One could think of training solely on molecules the same way mol-CLR [13] did, although for this task the authors used 10 millions of unlabeled molecules.

5.2.3 Data augmentation. Graph molecules augmentations have been proposed in mol-ICLR [13]. Regarding language processing, we could have trimmed the sequence lengths and inverted a few words. We're unsure if this would have helped or not but it could be a future idea to be explore (CLIP [10] insists on its benefits for contrastive learning).

5.2.4 Ensembling. Instead of averaging cosine similarity results from several models, we could concatenate all embeddings from all models and perform cosine similarity on these. Another idea would be to process all these different embeddings with an additional "decision" network which could be trained to merge various models predictions.

5.2.5 Graph convolutions. Using the basic graph convolution operator [9] may not be powerful enough to distinguish complex molecule structures as the aggregation performs a mean of all its neighbors and may loose the ability of counting. GIN [14] proposes to replace the mean by a sum to have more discriminating power regarding graph patterns. *For example, a carbon atom with 2 hydrogen and 2 oxygen neighbors will get the same convolution output if it had only a single hydrogen and a single oxygen neighbor.* A deeper study of the molecules dataset would be useful to see if there are truly graphs which show this ambiguity and if the Mol-2-Vec [7] prepossessing did not remove most of these ambiguities.

6 CONCLUSION

During this project, we faced the difficult problem of contrastive learning with the additional burden of complex data structures (natural language sentences and sparse graphs stuctures). By iteratively exploring the vast amount of combinations of hyper parameters, architectures and loss functions (more than a hundred experiments with at least half a day of training each), we trained seven equally accurate models. By ensembling, these models achieved together a final LRAP score of 0.905. With an engineering effort as significant as the amount of time spent experimenting on the machine learning part of the project, we came up with a solution which can retrieve molecules from a text query.

REFERENCES

[1] Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A Pretrained Language Model for Scientific Text. arXiv:1903.10676 [cs.CL]

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. 2021. Emerging Properties in Self-Supervised Vision Transformers. arXiv:2104.14294 [cs.CV]

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709 [cs.LG]

[4] Xinlei Chen and Kaiming He. 2020. Exploring Simple Siamese Representation Learning. arXiv:2011.10566 [cs.CV]

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[6] Carl Edwards, ChengXiang Zhai, and Heng Ji. 2021. Text2Mol: Cross-Modal Molecule Retrieval with Natural Language Queries. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 595–607. https://doi.org/10.18653/v1/2021.emnlp-main.47

[7] Sabrina Jaeger, Simone Fulle, and Samo Turk. 2018. Mol2vec: Unsupervised Machine Learning Approach with Chemical Intuition. *Journal of Chemical Information and Modeling* 58, 1 (2018), 27–35. https://doi.org/10.1021/acs.jcim.7b00616 PMID: 29268609.

[8] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[9] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]

[10] 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. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV]

[11] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL]

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[13] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. 2022. Molecular contrastive learning of representations via graph neural networks. *Nature Machine Intelligence* 4, 3 (March 2022), 279–287. https://doi.org/10.1038/s42256-022-00447-x

[14] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? arXiv:1810.00826 [cs.LG]

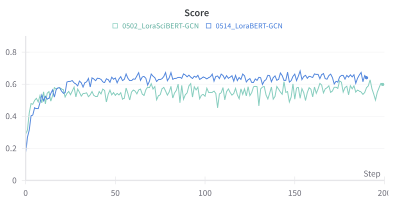


Figure 4: LRAP scores for experiments 502 and 514

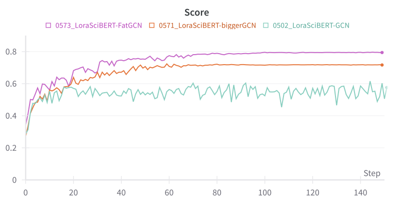


Figure 5: LRAP scores for experiments 502, 571 and 573

A APPENDIX: EXPERIMENTS

A.1 Results with LoRa

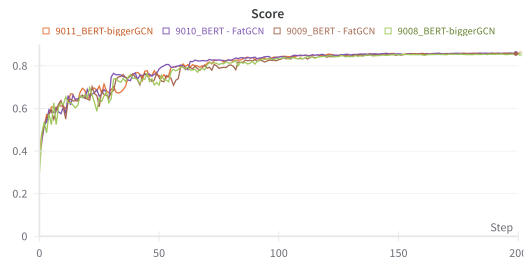
We here compare the results of experiments 502 and 514. Their validation scores are displayed on Figure 4. They were conducted with the same learning rate (3e-4) without a learning rate scheduler. Their specific parameters are detailed in Table 4. Memory restrictions keep us from increasing the batch size to 128 in experiment 502.

A.2 Results for the experiments with different GCN architectures

We compared the influence of the depth of the GCN architecture on several very close models: 502 (base GCN), 571 (big GCN),573 (fat GCN). All elements except for the GCN and the learning scheduler are identical between these experiments. 502 has no learning scheduler which explains the oscillations. We have reason to believe that the Base GCN would have still underperformed compared to 571 and 573 even with the same scheduler. The architecture and parameters of the models are detailed in Table 5 and their LRAP scores are displayed on Figure 5

A.3 Results for the best models trained with the contrastive loss

Out of all the experiments we led in the first part of our study with the initial contrastive loss provided with the baseline for this project, these 4 experiments are the ones that yielded the best LRAP scores. They were trained only using DistilBert in order to increase the batch size as much as possible and without LoRa which we did not feel helped us increase the batch size enough (complete architecture and parameters in Table 6). All these experiments were performed using the same learning rate scheduler in "plateau" mode: initial learning rate of $3e-5$, "patience" of 8 epochs and decreasing factor of 0.8. We can see here that increasing the batch size can make up for a smaller GCN architecture and vice-versa and we obtained very similar results with all these experiments. Their scores are visible in Figure 6.



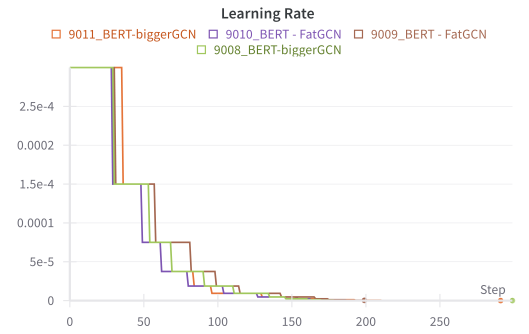


Figure 6: Score for various successful experiments trained with the contrastive loss

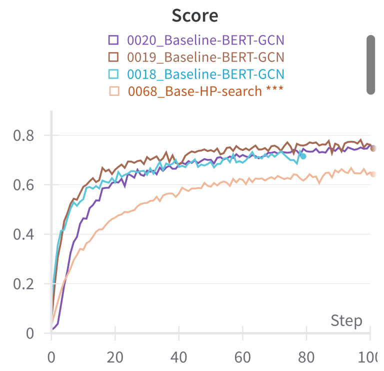


Figure 7: LRAP score for baseline experiments

A.4 Results for experiments trained with the binary contrastive loss

A.4.1 Baseline experiments. We trained the baseline experiments (BERT encoder and basic 3 layer GCN) with this new binary contrastive loss.Their detailed parameters can be found in Table 7. We used the learning rate 1e-4 proposed in [6]. We observe that the results (displayed on Figure 7) are much higher than the previous best result on the baseline (experiment 68, LRAP score of 66%).

Experiment ID	Batch size	Model size	LLM	GNN	LRAP
502	64	1.9M	LoRA SciBERT	Base GCN	61.6%
514	128	888k	LoRA BERT	Base GCN	71.8%

Table 4: Baseline experiments with LoRA BERT and SciBERT

Experiment ID	Batch size	Model size	LLM	GNN	LRAP
502	64	1.9M	LoRA SciBERT	Base GCN 3 layers	61.6%
571	64	2.9M	LoRA SciBERT	Big GCN 5 layers	71.6%
573	64	3.4M	LoRA SciBERT	Fat GCN 7 layers	79.6%

Table 5: Experiments with LoRa SciBERT and various GCN models

Experiment ID	Model size	Batch size	GNN	LRAP
9008	67.9M	164	Bigger GCN 5 layers	85.6%
9009	68.4M	128	Fat GCN 7 layers	85.9%
9010	68.4M	144	Fat GCN 7 layers	86.1%
9011	67.9M	180	Big GCN 5 layers	86.1%

Table 6: Successful experiments trained with the contrastive loss

Experiment ID	Model size	Batch size	Loss	LRAP
18	67.0M	32	Binary contrastive	71.5%
19	67.0M	64	Binary contrastive	79.9%
20	67.0M	128	Binary contrastive	77%
68	67.0M	128	Contrastive	66.7%

Table 7: Baseline experiments with different training losses

A.4.2 Other successful experiments. In order to get other models with high scores, we reproduced models with an architecture similar to those of experiments 9008-9011 and trained them with the binary contrastive loss. These architectures are presented in further detail in Table 8.Thanks to this loss, we were able to achieve similar results with smaller batches during training. We retained the learning scheduler (initial $lr = 1e - 4$) and DistilBERT encoder and played with the graph and the batch sizes to obtain high performances. The scores of all these experiments are displayed on Figure 8.

Freezing the LLM, training a large GCN from scratch

We reused the trained model 573 which we already presented (Lora Scibert model trained with a batch size of 64, a learning rate scheduler and the 'Fat GCN' architecture) in experiment 611. We froze the trained LLM obtained via 573 and trained 611 with a bigger batch size. The details about these experiments can be found in Table 9 and their respective scores are displayed on Figure 9. This technique enabled us to improve the LRAP score of 80% we had already obtained when training experiment 573.

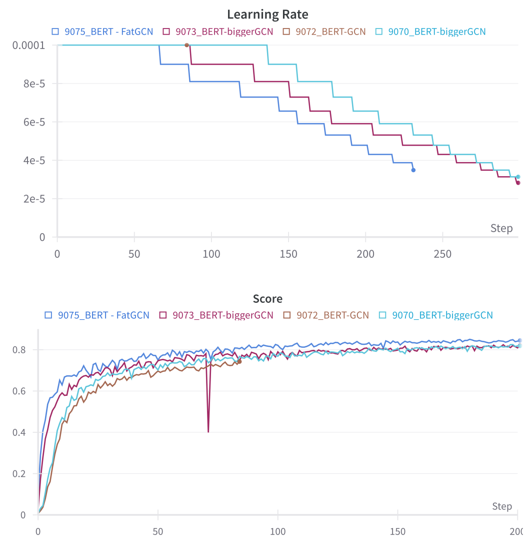


Figure 8: Score and learning rate for the best experiments with various GCN and batch sizes trained with the binary contrastive loss

Experiment ID	Model size	Batch size	GNN	LRAP
9070	67.9M	128	Bigger GCN 5 layers	83.6%
9072	67.0M	128	Base GCN 3 layers	79.9%
9073	67.9M	64	Bigger GCN 5 layers	83.0%
9075	68.4M	64	Fat GCN 7 layers	86.1%
9077	68.4M	92	Fat GCN 7 layers	86.3%

Table 8: Experiments with binary contrastive training loss

Experiment ID	Model size	Batch size	GNN	LRAP
573	3.4M	64	Fat GCN 7 layers	79.6%
611	2.4	192	Fat GCN 7 layers	86.7

Table 9: Fully trained experiment and experiment with the frozen LLM and GCN trained from scratch

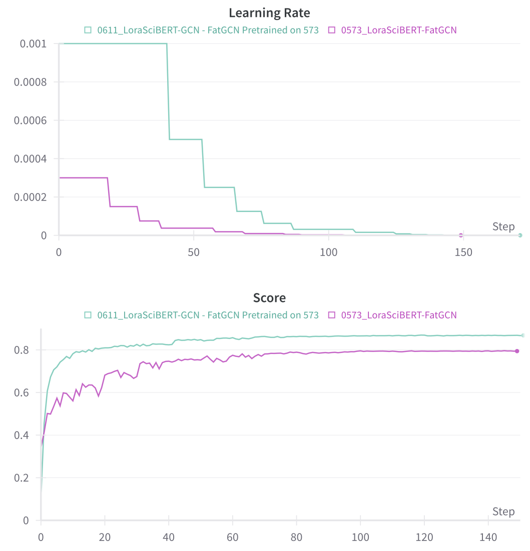


Figure 9: Score and learning rate for fully trained experiment 573 and experiment 611 with frozen LLM

B APPENDIX: TRAINING FRAMEWORK

This section describes the work which was done at software level which is definitely not a part that can be neglected when working in group on such a complex machine learning project (a lot of heavy computation power required, long training times for limited chances for improvements). We describe the constraints we had to face and the solutions we found to overcome them. We trained nearly a hundred experiments which requires good organization and a way to track and monitor everything. We came out with a convenient solution which allows respecting all the project constraints and allowed us to focus on the machine learning part of the project.

Constraints statement

As this work was to be done in group, we had to develop a framework to guarantee that everyone could work and share their code independently.

- Heterogeneous environments: several operating systems (linux, windows, macOS), various platforms (local laptop training, Kaggle Kernels notebook, remote machines at Ecole Polytechnique). On every platform, data has to be acessible and experiments results have to be stored in a way they can be retrieved. We used the Kaggle dataset feature to host the raw dataset as well as the preprocessed data.
- Privacy: as not sharing code was among the rules of the challenge, our source code remained private on GitHub (*which made cloning operations even trickier when using Kaggle kernels*).
- Reproducibility: all our experiments are reproducible (source code tracking under git, local and cloud storage of experiments results using Weights and biases).
- Cost: project total budget limit to 50euros. *Google Colab would not have met this requirement considering the amount of experiments we did and the long training times.*

We wrote a framework which takes and solves all these constraints at once and allows to focus on training models.

Definition of an experiment

An experiment is defined by a unique identifier and the instantiation of a model (tokenization method, architecture of the LLM and GNN), the configuration of an optimizer and training hyper parameters. At inference time, we're using this unique identifier so we can safely instantiate a network and reload the weights (*trained model are by construction compatible with inference. This avoids the risk of having a '.pth' weight file without knowing which architecture to use to reload it.*). Launching the training of an experiment goes seamlessly with the following command line, below is the example for experiment 620:

```
python train.py -e 620
```

The same exact training can be ran directly on Kaggle using their API.

```
python remote_training.py -e 620 -p -nb mol-nlp
```

Finally, the evaluation mechanism generates the submission csv file and the right Kaggle API command line which can be used to submit to the hosted challenge (with a message which allows to know exactly which experiment was submitted in which conditions).

```
python evaluation.py -e 620
```

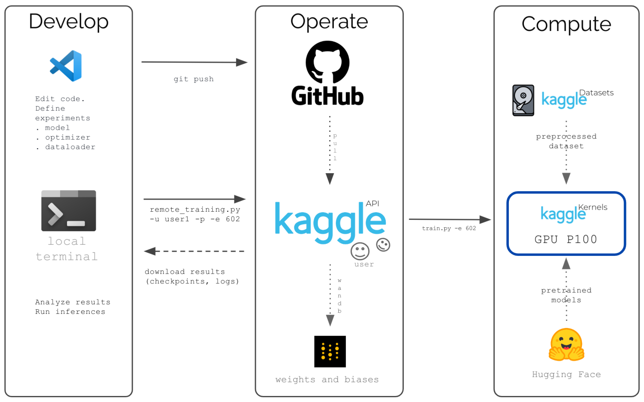


Figure 10: Remote training framework allows accessing free GPU resources on Kaggle Kernels. Local code is synchronized through our private Github repository (until the 4th of February) so the remote notebook can be run with the same code. The Kaggle dataset feature is used to store the raw dataset and the preprocessed data (for each tokenizer, we have stored the preprocessed dataset which saves nearly 10 minutes of computation time at the begining of each experiment). To fine tune our own models, we use Hugging Face as an exchange platform so we could easily start from a checkpoint from any remove computer. The Weights and biases platform is used to track the experiments results and monitor the training. Once finished, trained models can be retrieved by pulling the results. Note that the small piece of code to help access Kaggle notebooks freely from the terminal was made available publicly (totally independantly from this challenge) to be later used on other projects.

Training conditions

To setup the training loop, we started on a single NVIDIA GeForce RTX T500 local GPU with 4Gb of RAM (*the baseline provided by the challenge organizers did not even run because of memory limitation*). This was enough to make sure we could train, track and monitor progress of all our experiments and build a whole local training framework. We started by freezing the LLM weights to reduce the amount of memory required and get decent batch sizes even on the tiny T500 GPU. Due to low performances, the need to access bigger computation resources quickly came so we had to overcome this difficulty by making our framework agnostic to the training platform and still be able to recover our results and monitor from anywhere. The diversity of training and hardware platforms is shown in 10. We considered using Google Colab Pro (datasets and models shared through Google drive) but with the consideration that we have long training times, the cost would have been prohibitive. We found a good compromise and did many experiments using Kaggle Kernels through the API which are free and provide GPU access with 16Gb

Feature	**Nvidia T500**	**Nvidia RTX 2060**	**Nvidia K100**	**Nvidia A4000**
OS	Linux	Linux WSL	Linux in Docker	Linux
Location	local laptop	local laptop	Kaggle	Ecole Polytechnique
Access	direct	direct	Kaggle Kernels	SSH
Dataset access	local SSD	local SSD	Kaggle dataset	remote download + SSD drives
Memory	4 GB	6 GB	16 GB	24 GB
Student cost	-	Electricity $\approx$ +20 euros/month	Free	Free
Availability	∞	∞	30 hours/week 12hours/experiment	$\approx \infty$ weekends and night

Table 10: Comparison of the training platform and GPUs which were used during training

of RAM during a maximum time of 12 hours per session and up to 30hours per week.

C APPENDIX: STATE OF THE ART

In this supplementary section, we describe relevant works related to contrastive learning, graphs neural networks and the language models we've used.

C.0.1 Contrastive learning. The CLIP [10] paper introduced by OpenAI builds image representation embedded in the same vector space as textual representations. This is a multi modality contrastive learning task, CLIP seems to rely on the fact that contrastive learning requires very large batch sizes (32000 image/text pairs per batch according to the authors) and that augmentations are mandatory.

Mol-CLR [13] tackles the problem of learning powerful representations solely from molecules only. Recent papers (such as SIMCLR[3], Sim-SIAM [4] and DINO[2]) showed that great properties emerged from contrastive learning on images when applied to downstream tasks. So what about molecules? To be able to perform contrastive learning on a single modality, one needs to be able to augment the same data sample in several ways so that a human could still tell they're alike (in images, performing various crops or color transforms will still allow to recognize that the two augmented versions depict the same object for instance). MolCLR introduced interesting ways for molecule augmentations:

atom masking: vector representation set to a fixed token so the atom is considered masked / undefined)

bound deletion: removed a certain amount of edges across the graphs

subgraph removal: mask an atom and it k-hop neighbors until it saturates to a certain amount of masked atoms (and finally remove the bounds inside the selected sub graph)

This technique could be thought as a pre-training task for the GCN although it seems to require a huge amount of data as the authors using approximately 10 millions of unique unlabelled molecules.

C.0.2 BERT: a versatile language model. The BERT paper [5] tackles the masked language modeling problem by training the encoder part of the Transformer [12] model. BERT is a general purpose language representation model which become useful after fine tuning on downstream tasks (such as text classification or sentiment analyzis but not chatbots). On the contrary to causal language modeling, BERT is not trained using next word prediction (unlike the GPT family) but instead uses the whole sentence to predict a masked word in the middle of it. BERT is able to build great language understanding capabilities which is what we're looking for here: a downstream task in the chemistry domain. Researchers at Hugging Face [11] worked on a distilled version of BERT which results in smaller model with nearly as good language modeling capabilities (*97% as good, 60% of the original size, 60% faster*). Student model (DistilBERT) is trained to mimic the teacher (BERT)'s probability distribution for each output word. This is different from the original masked language training which supervises the network by telling if the predicted word matches with the ground truth word. Sci-BERT[1] is a BERT architecture which has been trained on a large scientific corpus made of more than one million scientific papers for a total of more than 3 billions of tokens (similar to BERT) and where the corpus has been tokenized. The tokenizer named SciVocab is provided along with the models on Hugging Face.

C.0.3 GCN: graph convolution networks. The pioneering and popular work of Kipf and Welling [9] has defined a simple yet powerful local graph convolution operation which becomes extremely useful when stacked in a neural network architecture. To each input node of the graph, we attach a vector. Each node vector is processed using linear projection layers followed by non linearity function (such as ReLu), just like a MLP would process each nod vector separately. The graph structure is then exploited by averaging local neighbor node content (*message passing*). Stacking several such operations allows extracting relevant graph structures. The processed graph output can either be used to classify each individual node (for nodes classification) or treating the graph as a whole by using an extra pooling layer (graph classification). For the molecules, we don't need to use the representation of each atom separately so a pooling mechanism will be used. We heavily rely on the Pytorch geometry GCNConv operator which is the combination of a linear layer (mix channels) and the aggregation layer (average neighboring nodes). The symmetrically normalized adjacency matrix is used and allows numerical stability (we have not faced any issue when stacking these layers). One important point though is that the mean operator used to aggregate neighboring content may break the "counting" capability. For example, *assume we have (H,H, O,O) in a neighborhood of an atom, the result of the convolution will be the same as the one with (H, O) neighbors*). We may have to keep in mind that the local structure may have been included a bit in the vector content thanks to the Mol-2-Vec [7] of each node... the previous counter example may not be not such a problem. Additionally, it seems like there's a recent line of research (GIN [14] - graph isomorphism networks) which may be able to distinguish structures better - *roughly speaking the average aggregation is replaced by a sum operator.*