In this report, we outline the main methodologies used in the ALTEGRAD challenge. The challenge consisted of retrieving molecules from a given list of graphs, representing molecules, based solely on a textual query without any additional reference or textual information about the molecules.

1 Architecture design

1.1 Text encoder

We assumed that selecting the text encoder architecture would be straightforward, favoring a transformer model such as BERT [1]. Our initial approach consisted of looking for a powerful model to encode text trained on massive amounts of data. We thought of phi2, BERT, DistilBERT [2]. We also searched for pre-trained models on Hugging Face that were trained with text data similar to ours such as ChemGPT [3] and BioBERT [4].
We found that the choice in the initial model had minimal influence on our model's ultimate performance. We attribute this finding to the highly specific nature of the provided dataset, which includes an extensive array of molecule names. Consequently, all tokenizers from Hugging Face proved inadequate, consistently dividing molecule names into numerous fragments. Furthermore, our model only utilizes the initial embedding of the original model output, thus not fully leveraging the benefits of pre-trained weights.
We tried to mitiGATe that choice by taking the average latent representation but it suffered from vanishing gradient.
We opted to develop and train our tokenizer and proceeded to train a DistilBERT model from scratch on top of it with masked language modeling.

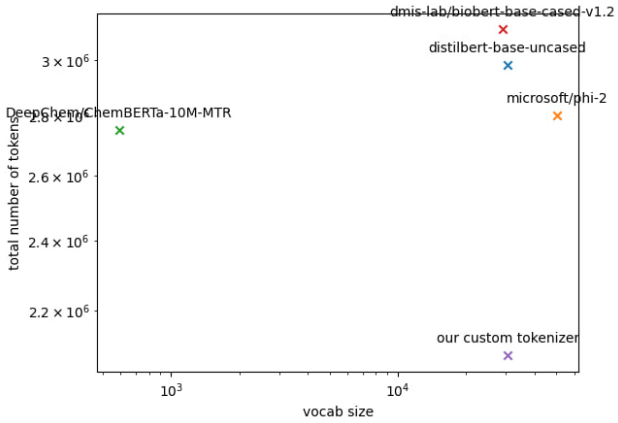


Figure 1: Total number of tokens in the entire dataset

Examples of tokenized sentence:

- Original sentence: UDP-alpha-D-galactofuranose(2-) is a UDP-D-galactofuranose(2-) in which the anomeric centre of the galactofuranose moiety has alpha-configuration. It is a conjuGATe base of an UDP-alpha-D-galactofuranose.
- BERT tokenizer (79 tokens):

```
['ud', '##p', '-', 'alpha', '-', 'd', '-', 'gala', '##ct', '##of', '##ura', '##nos',
'##e', ...]
```

- Our custom tokenizer (51 tokens):

```
['udp', '-', 'alpha', '-', 'd', '-', 'galactofuranose', ...]
```

1.2 Graph encoder

Graph Attention Networks (GAT) [5] represent a powerful class of neural network architectures designed specifically for tasks involving graph-structured data. Unlike traditional graph neural networks that typically aggreGATe information from neighboring nodes using fixed weights from the adjacency matrix, GAT introduce the concept of attention mechanisms, allowing the model to dynamically weigh the importance of different nodes during message passing. This enables GAT to selectively focus on relevant nodes while aggreGATing information, leading to improved performance in tasks requiring complex relational reasoning. Thus, we opted for GAT architectures to harness their capabilities in effectively handling the inherent complexities of the data and optimizing performance in the competition's context.

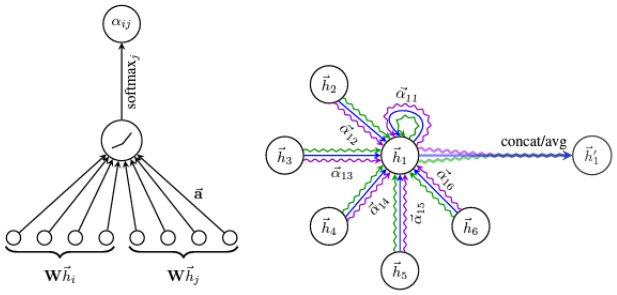


Figure 2: Graph attention network [5]

2 Training setting

2.1 Learning rate scheduler

The learning rate scheduler is a critical component in the training of neural networks as it dynamically adjusts the learning rate throughout training. This adjustment is essential because the learning rate determines the step size taken during optimization, impacting the convergence and stability of the model. Decreasing the learning rate gradually throughout the training process is particularly important as it allows for more fine-grained adjustments towards the end of training when the optimization process approaches convergence.

- **Exponential Decay:** We implemented a strategy where the learning rate is divided by a factor α after k epochs without observing any improvement in the LRAP metric. This approach aims to gradually decrease the learning rate as training progresses whenever it plateaus.
- **Cosine Scheduler:** Another technique we employed involved using a cosine scheduler, which operates within a specified range defined by η_{min} and η_{max}. This scheduler adjusts the learning rate according to a cosine function, gradually reducing it from η_{max} to η_{min} over the course of training. This method aims to provide a smoother and more controlled descent towards the optimal solution, potentially enhancing convergence and generalization performance. The cosine scheduler was found to be better than exponential decay.
- **Warmup:** We incorporated a two-epoch warmup period for the learning rate. This practice involves gradually increasing the learning rate during the initial epochs of training before transitioning to the scheduled learning rate adjustment. We adopted this approach to mitiGATe the potential issue of the model getting stuck in suboptimal solutions early in training due to a high learning rate.

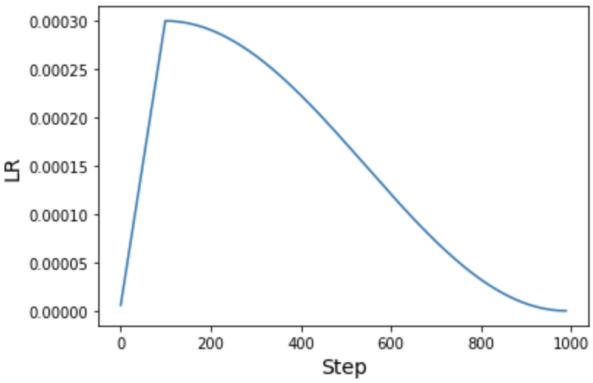


Figure 3: Cosine learning rate scheduler with warmup

2.2 Batch size and online hard sample mining

Our experiments revealed that employing a larger batch size frequently enhances the training of our neural network as stated in [6]. We believe that this is due to the increased difficulty for the network to differentiate between molecules when more samples are present in a single batch. Additionally, larger batches elevate the likelihood of encountering challenging samples, which is advantageous for training purposes.
To amplify this effect while staying within our computational constraints, we implemented online hard sample mining [7]. This involved calculating the loss for a larger batch and retaining only those samples with higher loss values, thereby prioritizing the samples that have the greatest impact on the backpropaGATion update. Just like the learning rate scheduler, we assumed that hard sample mining makes more sense later in the training process so we used a linear scheduler for it which improved the LRAP metric.

3 Graph augmentation

3.1 Data augmentation

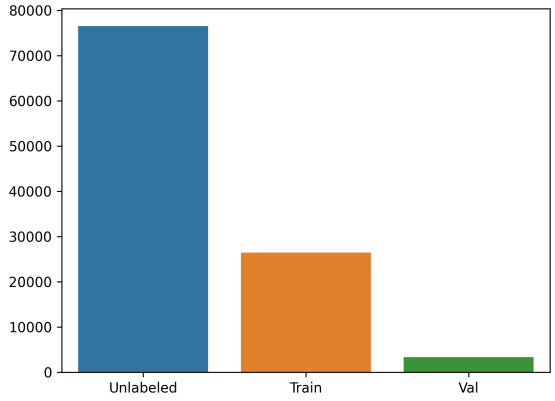
We employed data augmentation techniques [8] for graphs to address the challenges posed by limited training data. By augmenting the graph data with variations and transformations, we aimed to enrich the diversity of the training set and enhance the model's ability to generalize to unseen instances. Additionally, data augmentation helps mitiGATe overfitting by exposing the model to a wider range of graph structures and patterns, thus improving its robustness and performance on real-world data.

- **Edge Perturbation:** We perturbed the edge index, resulting in a modified adjacency matrix A' obtained by applying the XOR operation between the original adjacency matrix A and a matrix E composed of Bernoulli random variables.
- **Graph Sampling:** This augmentation involved randomly sampling nodes from the graph and extracting the corresponding subgraph.
- **K-hop Subgraph:** Similar to graph sampling, but ensuring that the extracted subgraph consists of connected components within a k-hop radius from the sampled nodes.
- **Features Noise:** We introduced Gaussian noise to the node features, enriching the variability of feature representations.
- **Features Shuffling:** Node features were randomly shuffled to introduce variability in their order within the graph.
- **Features Masking:** We masked certain node features, effectively obscuring specific information during training.

Features shuffling, despite its potential to introduce variability, ultimately led to a degradation in performance. Conversely, graph sampling emerged as a stabilizing factor for metrics during validation, yet it did not significantly enhance overall performance. Edge perturbation exhibited promising results, showcasing an improvement in performance metrics. Due to time constraints, we were unable to conduct a thorough exploration of noise, masking, and k-hop augmentation techniques.

3.2 Self-supervised pretraining

Due to the huge disparity in the amount of data between labeled and unlabelled data, see Figure [4], we try to use self-supervised learning to leverage this unused knowledge.



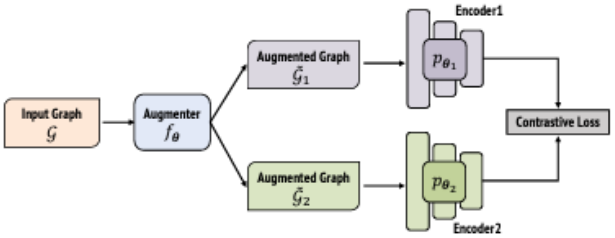


Figure 4: Data distribution (left), Self-supervised contrastive learning for graph encoder [8] (right)

The original framework that we used was proposed in the paper [8], it proposes the classical framework shown in Figure [4]. However, for its technical implementation, we highly rely on ideas from the well-known self-supervised framework for computer vision Ibot [9].
To describe it briefly here is how the framework is learning:

1. Initialize two networks using the same architecture, namely the student network and the teacher network. The student network will be the one through which the gradient will flow whereas the teacher network will be updated using the weights of the student network and performing an exponential moving average (EMA).
2. From a given graph x, transform it into two different views u, v using data augmentation techniques illustrated before.
3. Using these two views, we generate four graph embeddings using the two networks, we get h_u^s, h_v^s, h_u^t, h_v^t.
4. Using those four embeddings we guide through gradient updates the student network to output embedding close to the one of the teacher network. The main part is that the loss is computed between the embedding of the view u for one network and the view v for the second, thereby encouraging the network to overcome random data augmentations.

To enhance stability and overall performance, the Ibot framework incorporates several hyper-parameters, including two temperature parameters that regulate the sharpness of both the student and teacher networks. Additionally, it introduces a center variable, which represents the exponential moving average of the embeddings throughout the training process and is employed to center the teacher embedding before computing the loss.
The loss function for training such a network is typically a cross-entropy function, which in this case is adjusted to accommodate the new hyper-parameters:

$$\ell(s, t, center, \tau_s, \tau_t) = -(\text{softmax}((t - center)/\tau_t) * \log(\text{softmax}(s/\tau_s))) \quad (1)$$

While this framework was successfully trained and converged for our specific use case, the results did not meet our expectations. It only marginally increased the convergence rate of training on labeled data, and the performance did not improve significantly enough to be noticeable.

4 Experiments

4.1 Basic setting

The fundamental training configurations for our experiments were established as follows: we utilized a global GitHub repository for facilitating collaboration and a Weights & Biases project to monitor various metrics throughout the runs, encompassing the rate of convergence and overall performance.

In the subsequent table, you'll find a summary of each experiment we conducted along with the results on the training set and the performance on the validation set.

Experiment	Result on Validation
Increasing the number of epochs of the baseline training (10 epochs)	0.3480→0.4984
Normalizing the embedding before applying the loss	-
Using max or mean pooling for the NLP embedding instead of indexing	0.4984→0.397
Using a GAT architecture for the GNN	0.4984→0.549
Fine-tuning the tokenizer and the LLM	0.549→0.5572
Using a cosine scheduler with warmup	0.5572→0.6377
Increasing the batch size (64→256)	0.6377→0.7518
Reverting to AdamW optimizer and adjusting weight decay	0.7518→0.8306
Using an exponential decay scheduler	-
Implementing hard sample mining	0.8306→0.8515
Edge perturbation augmentation	0.8515→0.8616
Ensembling	0.8616→0.8801

Table 1: Results on Training and Validation Sets

A concise representation of the training of these various runs is provided in Figure 5:

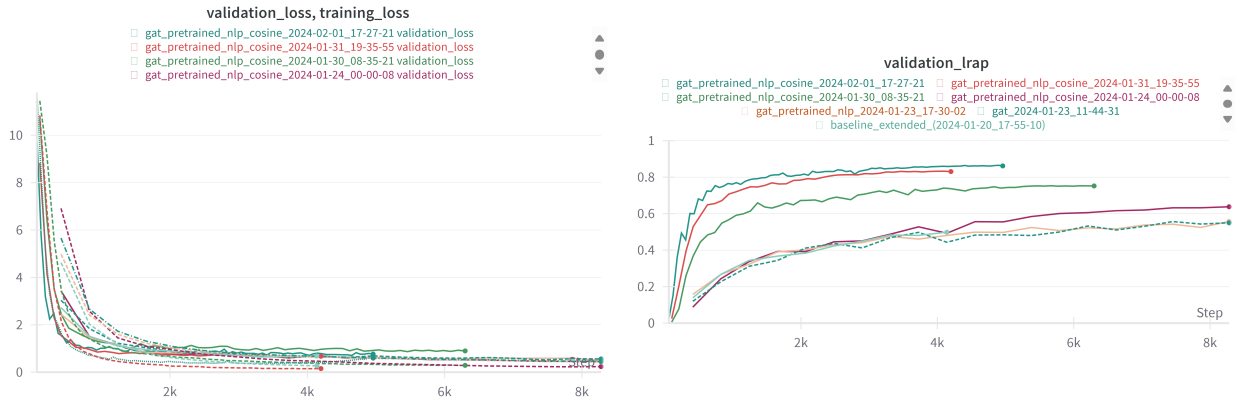


Figure 5: Different experiments: Training and Validation loss (left), LARP on the validation set (right) during training

Due to time and hardware constraints, we were unable to conduct any sweeps or hyper-parameter tuning. Therefore, all hyper-parameters were manually selected based on prior knowledge and a few evaluation runs. Another noteworthy observation that helps us determine the optimal settings is that our LARP on the validation set has consistently aligned closely with the results obtained on the testing set of Kaggle. This instills confidence that any enhancements we make in our experiments will be reflected on the leaderboard.

4.2 Results

Our final prediction utilizes an ensemble of two models, each employing the following configurations:

- NLP model: distilbert-base-uncased and its tokenizer first fine-tuned on the description corpus.
- GNN model: GAT architecture with 6 layers and a hidden dimension of 512, incorporating dropout in the GNN part (0.2).
- Optimizer: AdamW for 80 epochs with a batch size of 440 and a weight decay of 0.00002.
- The learning rate follows a scheduler with a cosine annealing starting from 1e-4 and ending at 1e-8, with warmup for the first two epochs.
- Hard sample mining with linear scaling starting at 440 (the batch size of the run) and ending at 180.
- Data augmentation for every data point, where the number of augmentations follows a Poisson distribution with a parameter of 0.3, capped at a maximum of 1, and only applying edge perturbation.

The disparity between the two models arises from one of them having its GNN pre-trained using the self-supervised method described earlier.

5 Further improvements

Now that we have addressed the intricacies of our implementation and explored various ideas and techniques during this competition, it's pertinent to consider avenues for further improvement that we did not have the opportunity to test and implement fully.

1. Firstly, to potentially amplify the efficacy of self-supervised pre-training, we could experiment with freezing the first layer of the Graph Neural Network (GNN) during training or for the initial epochs. This approach aims to safeguard the learned representations from the pre-training phase, preventing them from being overwritten or diminished during the early stages of training.

2. Secondly, we could delve deeper into the implementation of hard mining techniques by refining our data sampling strategies. Utilizing advanced samplers to assemble batches comprising similar text or graphs could intensify the focus on challenging instances during model training, potentially enhancing the model's ability to generalize across diverse and complex datasets.

3. Additionally, we tried to use the unlabelled graph during the training directly by adding them to the batch to add some difficulty to the training with more graph inputs than textual inputs, but we did not succeed to make this approach work. Nevertheless, exploring alternative methods for leveraging unused graph data remains a promising avenue for future investigation.

4. Furthermore, augmenting textual data with techniques like sentence splitting, word permutation, and synonym substitution could further diversify our training corpus, fostering robustness and adaptability in our model's language understanding capabilities.

5. Finally, transitioning to the Bayesian Personalized Ranking (BPR) loss function [10] could refine our optimization strategy, aligning it more closely with the Label Ranking Average Precision (LRAP) metric and potentially yielding more accurate and meaningful results.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

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

[3] Nathan Frey, Ryan Soklaski, Samuel Axelrod, Siddharth Samsi, Rafael Gomez-Bombarelli, Connor Coley, and et al. Neural scaling of deep chemical models. *ChemRxiv*, 2022. This content is a preprint and has not been peer-reviewed.

[4] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[6] Luyu Gao and Yunyi Zhang. Scaling deep contrastive learning batch size with almost constant peak memory usage. *CoRR*, abs/2101.06983, 2021.

[7] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.

[8] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *ACM SIGKDD Explorations Newsletter*, 24(2):61–77, 2022.

[9] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *arXiv preprint arXiv:2111.07832*, 2021.

[10] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012.