Molecule Retrieval with Natural Language Queries : Data challenge report

Sacha Braun
sacha.braun@polytechnique.edu
École Polytechnique
Palaiseau, France

Abstract

In the field of cheminformatics, accurately matching molecules with their textual descriptions is a challenging yet essential task. This project aims to address this challenge by developing advanced models for molecule retrieval using natural language queries. We introduce two distinct approaches to tackle this problem. The first approach employs a class of deep learning models, which effectively deciphers the complex relationships between molecular structures and their textual descriptions, achieving an LRAP score of 0.90. Building on this class of models, our second approach incorporates boosting strategies to refine the model's performance further. This method significantly enhances precision, achieving a LRAP score of approximately 0.9480.

1 Introduction

The intersection of natural language processing (NLP) and molecular science is a novel area of exploration that presents unique challenges. This project aims to develop a method for retrieving molecular structures, represented as graphs, using natural language queries. The complexity of this task arises from the fundamental differences in information representation between textual descriptions and molecular graphs. To bridge this gap, we employ machine learning techniques, specifically contrastive learning, (Weng 2021) to co-train a text encoder and a molecule encoder. This approach enables the identification of corresponding molecules for given text queries by aligning similar text-molecule pairs closer in a shared representation space, despite the absence of direct textual information about the molecules.

Evaluating our models' performance through the Label Ranking Average Precision (LRAP) score allows us to quantify their effectiveness in accurately matching molecules to text queries. Achieving a high LRAP score is indicative of a model's capability to navigate the complex relationship between the structured, often qualitative knowledge of language, and the precise, quantitative properties encoded in molecular graphs.

MVA 2023-2024, Febuary 2024, Paris, FR

Contrastive Learning is a powerful technique in machine learning that aims to learn effective representations by contrasting similar (positive) and dissimilar (negative) pairs of data points. At the heart of this approach is the concept of contrastive loss (Chopra et al. 2005), which encourages the model to bring representations of similar items closer together while pushing apart those of dissimilar items. A specific instantiation of this idea is triplet loss (Schroff et al. 2015), where a model learns from triplets of data points consisting of an anchor, a positive example similar to the anchor, and a negative example dissimilar to the anchor. Another variant, the Lifted Structured Loss (Oh Song et al. 2016), extends this concept by considering the relationships among all pairs within a batch of data, thereby providing a more comprehensive learning signal. Boyd's BYOL (Grill et al. 2020) approach sidesteps the need for contrasting negative samples by using a dual-network system. This system refines representations through a self-supervised mechanism, establishing new benchmarks for learning efficiency and representation quality without traditional negative sample contrasts.

In this report, we present our work on the Molecule Retrieval Project (Michalis Vazirgiannis 2023), where we explore various facets of contrastive learning. We delve into the comparative impact of different loss functions, the efficacy of varied NLP models, and the implementation of contrastive learning's key components. Furthermore, we introduce a strategy inspired by boosting, aimed at refining the model's performance. This comprehensive approach underscores our effort to identify the most effective methodologies for improving the accuracy and efficiency of molecule retrieval.

2 Methodology

2.1 Dataset

Our dataset for the Molecule Retrieval Project comprises a collection of molecules and their corresponding textual descriptions. The dataset is structured into training, validation, and test sets, with 26,408 training samples, 3,301 validation samples, and 3,301 test samples. Additionally, it includes a token embedding dictionary mapping molecule substructure tokens to embeddings, and raw graph files detailing the edgelist and substructure tokens for each molecule. We generated our graph dataset using those token as provided

in the baseline, and we used the tokenizer associated to its corresponding text model to tokenize the text dataset.

2.2 Model

2.2.1 Graph model. The graph model begins with a flexible input layer, designed to accommodate a wide range of node feature sizes and utilizes a variety of convolutional techniques for processing molecular graphs. Through empirical investigation, we assessed the effects of different graph convolution architectures on model performance. Among the various models evaluated, we included:

- **GCNConv** (Kipf and Welling 2016) offers a straightforward approach to graph convolution, leveraging the graph structure directly to aggregate neighbor information.
- **SAGEConv** (Hamilton et al. 2017) extends this by sampling and aggregating neighbor features, optimizing computation and memory usage.
- **ChebConv** (Defferrard et al. 2016) uses Chebyshev polynomials to capture the graph's spectral properties, allowing for a broader capture of graph topology.
- **GINConv** (Xu et al. 2018) implements a more powerful form of neighborhood aggregation that can distinguish between different graph structures more effectively.
- **GATConv** (Velickovic et al. 2017) introduces attention mechanisms to graph convolution, enabling the model to focus on more important parts of the graph structure.
- **TransformerConv** (Shi et al. 2020) brings the power of transformer models to graph data, applying self-attention mechanisms to graph nodes.

To ensure stability and efficiency in training, the model employs BatchNorm after each convolutional layer. This technique normalizes the inputs to each layer, addressing the potential issue of internal covariate shift by standardizing the inputs to have zero mean and unit variance, thereby accelerating the learning process and allowing to use much higher learning rates (S. Ioffe 2015).

We also build the graph convolutional layers with a residual structure. The idea behind the residual structure is that the desired underlying mapping $\mathcal{H}(x)$ can also be approximated as the mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$. It has empirically been shown (K. He 2016) that it is easier to optimize the residual mapping than to optimize the original one as the learned residual functions have small responses in general.

In our exploration of molecular graph encoding, we experimented with three distinct pooling methods: global add pool, global max pool, and global mean pool (Nikolentzos and Vazirgiannis 2023). Global add pool sums up node features, capturing the overall presence of features within a molecule. Global max pool highlights the most prominent feature across nodes, emphasizing critical structural elements. Global mean pool, on the other hand, averages node features, providing a balanced representation of the molecule's characteristics. Concluding our model architecture, we incorporated three Multilayer Perceptron (MLP) layers, further refining the extracted features into a robust representation suitable for downstream tasks.

2.2.2 Text Model. In our project, we compared three text encoder models for processing scientific descriptions of molecules. SciBERT (Beltagy et al. 2019), though the heaviest, is known for its superior accuracy in scientific contexts. Distill-Bert (Sanh et al. 2019) offers a lighter, faster alternative, optimizing resource usage without significantly compromising performance. ChemBERTa (Chithrananda et al. 2020), stands out as the most lightweight model we found suitable for our scientific dataset. Following these models, we implemented a linear projection into the embedding space, complemented by layer normalization (Ba et al. 2016), and concluded the process by scaling the embeddings with a temperature factor to optimize matching accuracy.

2.3 The choice of the loss

2.3.1 Contrastive Loss. The first loss we used is the contrastive loss (Chopra et al. 2005), that was used in the baseline of the code. This method aims to encode input samples into embedding vectors, ensuring that samples from the same class are embedded closely together, while those from different classes are distinctly separated. This is achieved by minimizing the distance between embeddings of the same class and maximizing it for those of different classes.

2.3.2 Triplet Loss. Triplet loss (Schroff et al. 2015) is designed to improve the embeddings or representations of data points by utilizing a trio of data points: a graph embedding anchor (a reference point), its associated positive text embedding, and a well chosen negative text embedding. The objective is to arrange the embedding space so that the anchor is closer to the positive than to the negative. We related this loss with the cosine similarity that we will use later to match graph embeddings to their corresponding text embeddings. To do so, we choose the negative text embedding as being the negative embedding in the batch that maximizes the cosine similarity with the anchor.

The loss calculation can be summarized with the equation:

$$\mathcal{L}_{\text{triplet}}(x, x^+, x^-) = \max\left(0, \cos(f(x), f(x^-)) - \cos(f(x), f(x^+)) + \delta\right)$$

The loss becomes zero when the distance between the anchor and the positive is less than the distance to the negative by at least the margin δ, promoting a well-separated embedding space. The margin plays a crucial role by enhancing the separation between dissimilar points, aiding in the model's generalization. It also helps control overfitting by preventing the model from focusing excessively on individual examples.

However, setting a very large margin can complicate training, as satisfying such a constraint may be challenging in complex data spaces. As it may be complicated to train a model from scratch only on this loss, we used a loss a mixture of this loss, and the contrastive one, for small λ: $\mathcal{L} = \mathcal{L}_{\text{triplet}} + \lambda \mathcal{L}_{\text{contrastive}}$

2.3.3 Lifted Structured Loss. The Lifted Structured (Oh Song et al. 2016) Loss is employed to refine the embeddings of graphs and text by emphasizing correct pairings within a margin of separation. This loss function computes pairwise cosine distances, promoting closer distances for matching pairs and penalizing similar distances for non-matching pairs. It adjusts the embedding space to ensure that similar items are closer together, enforcing a margin of differentiation. The loss is then normalized by the batch size, ensuring a consistent scale across different batch sizes. This approach effectively structures the embedding space, enhancing the model's ability to discriminate between matching and non-matching pairs.

$$\mathcal{L}_{\text{lifted}} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{1}_{y_i = y_j} \left(\max \left(0, d_{ij} + \delta - \min_{k: y_k \neq y_i} d_{ik} \right) \right)^2$$

2.4 Training loop

2.4.1 Embedding on the same space. We also present other strategies that we implemented. To ensure coherence between graph and text encodings and to unify them within the same embedding space, we pursued two strategies. The first involved training a discriminator to ascertain if an embedding was derived from text, promoting indistinguishable encodings from both domains. The second strategy drew inspiration from VQ-VAE (Van Den Oord et al. 2017) methods, quantizing the continuous embeddings into a discrete space, thereby aligning the graph and text representations on the same set of features. This strategy allowed to stabilize the results and improved the scores we got using the contrastive loss, but was outperformed by the other two losses.

2.4.2 Ensure a large batch size. In contrastive learning, a large batch size is crucial as it ensures the presence of a diverse and comprehensive set of negative samples, which are essential for the model to learn distinctive features. A broad range of negatives provides the necessary complexity and challenge, enabling the model to develop robust representations that can effectively distinguish between different examples. Without a sufficiently large batch size, the variety of negative samples is limited, which can impede the learning process and the quality of the resulting embeddings. To accommodate larger batch sizes and enhance computational efficiency, we used a freezing decay strategy on the layers of each text model. We begin by using a small batch size and, after a specified number of epochs, freeze one layer to reduce GPU memory usage. This reduction in memory allows us to increase the batch size. We then fine-tuned our model using a new dataset that represents the text embeddings from the text model, thus enabling the exclusive use of the graph model and conserving computational resources. This approach is distinct from freezing all layers of the text model, as it eliminates the need to store the model's weights altogether. Such strategies have significantly enhanced our processing capabilities, in line with our goal of achieving coherent and unified embeddings. Additionally, we evaluated the model's performance in its float16 version to allow for an increased batch size; however, this method proved to be ineffective.

2.5 Boosting strategy

In our efforts to improve the Label Ranking Average Precision (LRAP) scores, we explored the application of boosting strategies, drawing upon the collective strengths of multiple predictors, denoted as h_i. The core idea was to capitalize on the individual capabilities of each predictor to contribute positively to the overall LRAP score. Inspired by traditional boosting techniques, our approach focused on optimizing the weighted sum of predictors, represented as

$$\text{LRAP}\left(\sum \alpha_i h_i(x)\right),$$

with the constraint that $\alpha_i \geq 0$.

In our exploration of strategies for selecting weights, we coded an AdaBoost optimizer and iterative binary searches. However, those strategies needed to build a csv file for the validation dataset, hence limiting the reproductibility of the experiment, while not providing outstanding results. The chosen strategy involved setting weights α_i proportional to the exponential of each model's validation LRAP score, $\alpha_i = e^{\text{val(LRAP)}}$, and the training of models able to catch different structures of the inputs. This approach also prioritizes models with superior validation performance, thereby enhancing the overall predictive accuracy.

To further enhance our model diversity and performance, we employed a genetic algorithm-inspired approach. This involved mutating a percentage of the weights in a finely tuned model and subsequently fine-tuning this new variant. This method aimed to probe a wider array of model configurations in search of optimal solutions.

3 Results

In this section, we summarize key findings from over 70 experiments, emphasizing the influence of hyperparameters and comparing various loss functions to showcase our most significant results. For detailed information on each specific parameter contributing to these results, please refer to the associated configuration files available in our GitHub repository[1]. We utilized TensorBoard to analyze the outcomes,

[1] https://github.com/ElSacho/NLP_Molecule_Retrieval

ensuring that all presented results can be found within the corresponding files in the *log* folder on GitHub.

Our experiments were primarily conducted over 200 epochs, utilizing the Adam optimizer (Kingma and Ba 2014) to investigate the influence of its key parameters on our findings. For the graph model, we employ three convolutional layers followed by a single-head GAT layer with a hidden dimension of 300, and the output is then processed through four MLP layers, resulting in a final output dimension of 300. Given the variety of loss functions tested and their differing values, we found that focusing solely on plots of the losses did not align with the objectives of our problem. Instead, we chose to examine the LRAP scores obtained on the validation dataset. We build our similarity matrix using the cosine similarity between the embeddings. While our analysis mainly fine-tuned the best scores from each architecture, we opted to present the results prior to fine-tuning. This approach provides a clearer hierarchy between the models. Generally, fine-tuning resulted in an approximate 0.02 absolute increase in the LRAP score on the validation dataset.

3.1 Graphs Model Results

For the convolutional graphs layers evaluated, performance was relatively consistent across the board, Fig 1 with a slight preference for the *cheb6* architecture, which outperformed others marginally. The *max*-pooling strategy outperfomed the two others Tab 1.

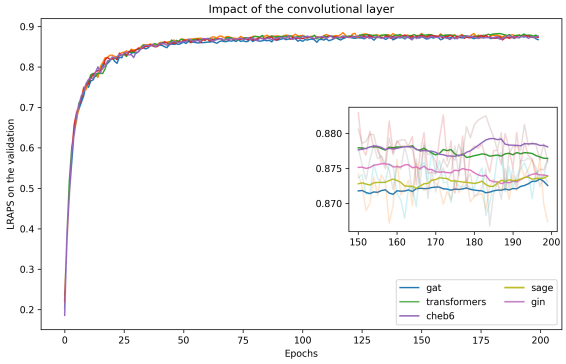

Figure 1. Impact of the convolutional architecture on the LRAPS on the validation dataset, with a zoom on the last 50 epochs

Model	LRAP Score
Add	0.8393
Mean	0.8832
Max	**0.8955**

Table 1. Performance of the pooling methods

3.2 Text Model Performance

Model	LRAP Score	Training time
SciBERT	**0.8825**	≈ 19h
Distill-Bert	0.8732	≈ 6h
ChemBERT	0.6103	≈ 5h

Table 2. Performance of Text Models

Among the text models, SciBERT achieved the highest LRAP score at 0.8825 Tab 4, closely followed by Distill-Bert with 0.8732. ChemBERT lagged significantly, registering a score of 0.6103, indicating a substantial performance gap for this task. However, it's important to note that SciBERT requires a longer training time compared to Distill-Bert, which suggests that the slight improvement in performance comes at the cost of increased computational resources.

3.3 Impact of the temperature

We found that the parameters does not significantly modify the results we obtained with our settings. Fixing this parameter to 0.8 seems to be slightly better Fig 2.

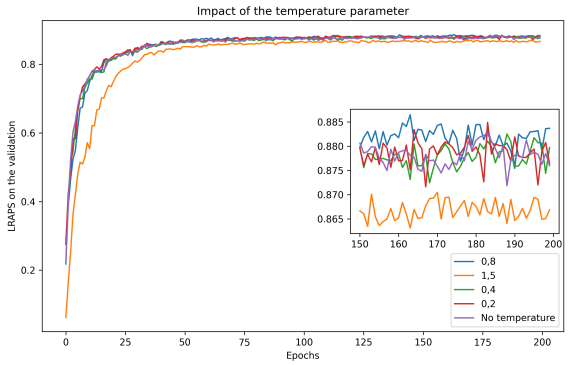

Figure 2. Impact of the temperature parameter on the LRAPS on the validation dataset, with a zoom on the last 50 epochs

3.4 Loss Function Comparison

Loss Type	LRAP Score
Contrastive	0.8521
Triplet	**0.8856**
Lifted	0.8793

Table 3. Comparison of Loss Functions

The comparison of loss functions revealed that the Triplet loss yielded the highest LRAP score at 0.8856, narrowly edging out the Lifted loss at 0.8793 Tab 3. The Contrastive loss scored 0.8521, positioning it as the least effective of the three for this particular task.

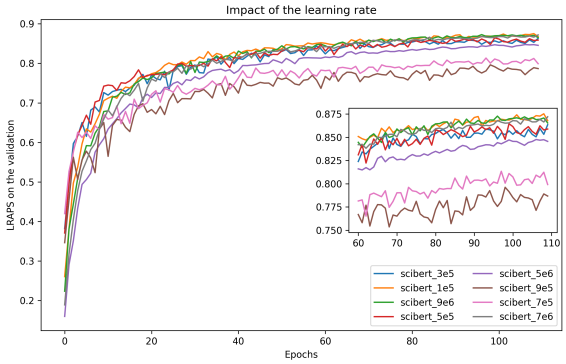

Figure 3. Impact of the learning rate on the LRAPS on the validation dataset, with a zoom on the last 50 epochs

3.5 Optimizer parameters

For the Adam optimizer, adjustments to the learning rate and betas were explored. No significant changes were observed with beta1 values ranging between 0.8 and 0.95. However, the optimal result was obtained with beta1 set to 0.92, achieving an LRAP score of 0.8834 Fig 3.

3.6 How to aggregate results

Among all the results we proposed, the best LRAP score on the validation dataset is approximately 0.90. We now show that summing several csv files can significantly improved the results. We try to derive a framework to empirically explain which models are better to be summing together. To do so, we proposed to show the results we get from aggregating several models based on the structure of their parameters. Following all this strategy, we reached a final LRAP score of approximately 0.9475.

We show that the choice of the model's parameters does impact the results. We start by exposing some insight on the impact of the model's parameters for the final results. The first results Tab. 4, leads to several conclusions : the learning rate with the same parameters does not catch many different structure compared to the use of a different convolution layer that leads to a significantly better results. Indeed, one can think that the same model with different learning may converge to the same predictor, whereas different convolutional layers leads to sigificantly different predictions. The same conclusion can also be made regarding the choice of the text model: despite a Distill-Bert model proposing lower results compared to Scibert, when aggregating those two results it leads to significantly better results.

We can also try to investigate to what extend a fine tuning (higher margin for the triplet loss, and higher batch size)

Table 4. Impact of the Architecture

Type of Models	Text Model	LRAP Score	# of Files
Learning Rates	Distill-Bert	0.9169	6
Learning Rates	Scibert	0.9220	6
Learning Rates	Both	**0.9361**	12
GNN	Distill-Bert	0.9317	5
GNN	Scibert	0.9377	5
GNN	Both	**0.9398**	10

leads to better results Tab 5. Our conclusion is that fine tuning a model significantly improve the embeddings.

Table 5. Impact of the Fine-tuning

Type of Models	Text Model	LRAP Score	# of Files
Before Fine Tuning	Scibert	0.9169	3
After Fine Tuning	Scibert	**0.9265**	3

We were also concerned by the impact of bad predictors on the final boosted one. We show Tab 6 that bad predictors, when combined with better ones, does not significantly impact the final score.

Table 6. Bad Predictors Do Not Impact the Final Score

Type of Models	Text Model	LRAP Score	# of Files
Best and Worst	Scibert	0.9363	6
Best Only	Scibert	0.9359	3

We conclude by noting that, surprisingly, neither the choice of loss function nor the earlier described mutation strategy significantly enhanced the results Tab 7. Ultimately, our carefully selected strategy achieved an LRAP score of 0.9460 on the validation dataset.

Table 7. Impact of Different Factors on Model Performance

Type of Models	Text Model	LRAP Score	# of Files
Losses	Distill-Bert	0.9198	6
Mutation	Scibert	0.8872	10
All the files	Both	**0.9460**	58

4 Conclusion and perspective

Our research involved extensive model comparisons, encompassing the trial of three distinct loss functions and the fine-tuning of numerous parameters. After conducting over a hundred experiments to identify the optimal hyperparameter mix, we established that SciBERT is ideally suited for this task. However, it's noteworthy that DistillBert, while marginally less effective, delivers comparable results in a significantly reduced timeframe. Our focus was primarily on

the batch size, a crucial aspect of contrastive learning. Future research could explore another key element of contrastive learning : the selection of negative samples. Our current method utilizes negative sample selection within individual batches, yet there exists potential for advanced techniques that could, for example, restructure the dataloader to cluster similar embeddings, possibly using spectral clustering on the post-training cosimilarity matrix.

Moreover, the boosting method we developed markedly enhanced performance, albeit with greater computational costs. The deployment of over fifty models, each undergoing extensive training and fine-tuning periods, resulted in substantial energy consumption, raising environmental concerns. Additionally, the varying performance of models across different result classes hints at the possibility of specialized approaches, such as Mixture of Experts, tailoring models to excel in specific domains.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676* (2019).

Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. 2020. ChemBERTa: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885* (2020).

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 1. IEEE, 539–546.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016).

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33 (2020), 21271–21284.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

S. Ren J. Sun. K. He, X. Zhang. 2016. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition* (2016).

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

Lutzeyer Johannes Guokan Shang Chatzianastasis Michail Giannis Nikolentzos Michalis Vazirgiannis, Abdine Hadi. 2023. Molecule Retrieval with Natural Language Queries. https://www.kaggle.com/competitions/altegrad-2023-data-challenge

Giannis Nikolentzos and Michalis Vazirgiannis. 2023. Deep Learning methods for Graphs and Sets. *MVA - Altegrad course* (2023).

Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. 2016. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4004–4012.

C. Szegedy. S. Ioffe. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International Conference on Machine Learning (ICML)* (2015).

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509* (2020).

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems* 30 (2017).

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* 1050, 20 (2017), 10–48550.

Lilian Weng. 2021. Contrastive Representation Learning. *lilianweng.github.io* (May 2021). https://lilianweng.github.io/posts/2021-05-31-contrastive/

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).