Augmenting Message Passing by Retrieving Similar Graphs

Dingmin Wang  
University of Oxford, UK  
dingmin.wang@cs.ox.ac.uk

Shengchao Liu*  
Mila, Canada  
liusheng@mila.quebec

Hanchen Wang†  
University of Cambridge, UK  
hw501@cam.ac.uk

Linfeng Song  
Tencent, United States  
lfsong@tencent.com

Jian Tang  
Mila - HEC Montréal, Canada  
jian.tang@hec.ca

Song Le  
BioMap and MBZUAI  
le.song@mbzuai.ca

Bernardo Cuenca Grau  
University of Oxford, UK  
bernardo.cuenca.grau@cs.ox.ac.uk

Qi Liu‡  
The University of Hong Kong  
qi.liu@cs.ox.ac.uk

ABSTRACT

Graph Neural Networks (GNNs) are effective tools for graph representation learning. Most GNNs rely on a recursive neighborhood aggregation scheme, named message passing. In this paper, motivated by the success of retrieval-based models, we propose a non-parametric scheme called GraphRetrieval, in which similar training graphs associated with their ground-truth labels are retrieved to be jointly utilized with the input graph representation to complete various graph-based predictive tasks. In particular, we take a well-trained model with its parameters fixed and then we add an adapter based on self-attention with only a few trainable parameters per task to explicitly learn the interaction between an input graph and its retrieved similar graphs. Our experiments on 12 different datasets involving different tasks (classification and regression) show that GraphRetrieval is able to achieve substantial improvements on all twelve datasets compared to three strong GNN baseline models. Our work demonstrates that GraphRetrieval is a promising augmentation for message passing.

1 INTRODUCTION

Graph neural networks (GNNs) are a class of neural architectures for supervised learning which have been successfully adopted in a wide range of applications involving graph-structured data, such as molecule classification [9, 24], recommendation systems [41], and knowledge graph completion [31]. Despite their success in applications, GNNs are also known to suffer important practical limitations. These include issues such as over-smoothing, [3] over-squashing [1], and example forgetting [34]. Furthermore, in contrast to other applications of deep learning such as NLP [7], graphs from different domains have diverse topologies and there are no pre-trained models that can provide universal graph representations.

In this paper, motivated by the success of retrieval-based models for NLP [11, 20, 22, 25, 38, 42], we show that the performance of GNN-based models can be significantly enhanced if the training data is also exploited at testing time for making predictions. This is in contrast to the standard setting in supervised learning, where the training data is typically discarded once training has been completed and the model relies solely on the learned parameters for making predictions.

We focus on graph classification and regression tasks using GNNs. Given a set of training examples consisting of graph-label pairs \((G_i, l_i)\), where nodes in the graphs are annotated with numeric feature vectors, our aim is to learn a model that predicts label \(l\) given an input graph \(G\). In this setting, a GNN with \(T\) layers computes on an input graph \(G\) via the following three steps:

1. **Message passing.** Each node receives messages from its neighbor nodes and updates its feature vectors by aggregating the vectors from its neighbors. This process is repeated in each of the layers in the GNN and hence the final vector assigned to a node will be influenced by the vectors of all nodes in its \(T\)-hop neighborhood.

2. **ReadOut.** Node-level representations are pooled (e.g. by means of max-pooling or mean-pooling) to provide a graph-level representation vector \(h_G\).

3. **Prediction.** Vector \(h_G\) is passed as input to another network (e.g., a multi-layer perceptron) to predict the output label.

We propose an effective and model-agnostic approach that exploits the training examples to make predictions in this setting. At training time, our approach consists of the following steps:

1. **Standard supervised training.** Initially, we train the GNN-based model of interest from examples in the usual way, and obtain a trained model \(M\) as a result.

2. **Indexing.** We apply \(M\) to each of the training examples \(\{(G_1, l_1), \ldots, (G_n, l_n)\}\) and obtain the corresponding graph-level representation vectors \(h_{G_1}, \ldots, h_{G_n}\). We then construct a key-value index having vectors \(h_{G_i}\) as the key and the corresponding example \((G_i, l_i)\) as value.

Given an input graph \(G\) at testing time, we compute its label \(l\) according to the following steps:

1. **Standard GNN application.** We apply \(M\) to \(G\) to obtain its representation vector \(h_G\) and its corresponding initial label \(l_G\) in the usual way.

2. **Graph retrieval.** We rank the representation vectors in the index according to their L2 similarity to \(h_G\), and retrieve the top-\(k\) vectors \(\{h_{G_{i_1}}, \ldots, h_{G_{i_k}}\}\) with highest similarity

*Second and third authors make equal contribution to this paper.

†Corresponding author.
score; we then retrieve the corresponding example graphs and labels \((G_1, l_1), \ldots, (G_n, l_n)\).

3) Self-attention. Inspired by the transformer proposed in [36], we exploit self-attention to compute the final output label \(l\) as a weighted sum of the initially predicted labels \(l_g\) and the labels \(l_1, \ldots, l_n\) of the retrieved training examples, where we denote the weights as \(w_0, \ldots, w_k\). To apply self-attention, we need a query and a set of key-value pairs [36]. To obtain the query vector \(q\) we multiply \(h_g\) by a learnable matrix \(W_1\); in turn, to obtain the keys (represented as column vectors in a matrix \(K\)), we multiply \(\{h_{G_1}, \ldots, h_{G_n}\}\) by a different learnable matrix \(W_2\). Finally, by applying self-attention to the query and the keys, we can obtain the weights \(w_0, \ldots, w_k\). The learnable matrices \(W_1, W_2\) can be optimised using the training data after the model \(M\) and the index have been obtained, and before the overall pipeline is applied to the testing set.

We have implemented our approach and used it to enhance three strong GNN baselines, GCN [19], GIN [39], and PNA [5]. We have then evaluated the performance of our approach in comparison to the baselines on a wide range of benchmarks, including two image datasets (MNIST and CIFAR10), eight molecule datasets (BBBP, Tox21, ToxCast, SIDER, ClinTox, MUV, HIV, and BACE), and two large-scale quantum chemistry datasets (PCQM4M and PCQM4Mv2). Our main contributions are summarized as follows:

- Our method achieves substantial improvements on all twelve datasets for all baselines.
- We systematically study the effect of retrieving graphs, and discovered a strong correlation between retrieval quality and downstream task performance.
- We perform extensive ablations to study the design choices of our model to justify their effectiveness.

2 BACKGROUND

In this section, we briefly recapitulate the basics of GNNs and formulate the relevant graph classification and regression tasks.

2.1 Message-passing GNNs

In the context of GNNs, we define a graph \(G\) as a tuple

\[(V, E, \{x_v\}_{v \in V}, \{x_e\}_{e \in E})\]

where \(V\) is a set of nodes with each node \(v \in V\) annotated with a numeric feature vector \(x_v\), and \(E\) is a set of (undirected) edges where each edge \(e \in E\) is also annotated with a feature vector \(x_e\). For instance, in a molecule classification scenario, nodes may represent molecules, edges represent bonds between molecules, and feature vectors encode information about atoms and bond types.

When given a graph \(G = (V, E, \{x_v\}_{v \in V}, \{x_e\}_{e \in E})\) as input, a message-passing GNN [10] with \(T\) layers updates each node feature vector throughout by aggregating messages from its neighbors. Formally, for each layer \(1 \leq t \leq T\) and each node \(v \in V\), the feature vector \(h_v^{(t)}\) for node \(v\) in layer \(t\) is computed as follows:

\[m_v^{(t)} = \sum_{u \in N(v)} M_{t-1}(h_u^{(t-1)}, h_e^{(t-1)}, x_{(u,v)})\]

\[h_v^{(t)} = U_t(h_v^{(t-1)}), m_v^{(t)}\]  

where \(h_v^{(0)} = x_v\) for each \(v \in V\), and where \(M_t\) and \(U_t\) represent the GNN’s message passing and update functions in the \(t\)-th layer, respectively. The concrete description of these functions is given in terms of the learnable parameters of the model. Finally, a ReadOut function is applied on the node feature vectors in the outermost layer \(t = T\) to obtain a graph-level representation vector:

\[h_G = \text{AGG}\left\{h_v^{(T)} \mid v \in G\right\}\]  

where AGG can be any permutation-invariant operation on multi-sets applied to a vector in a component-wise manner, such average, summation or attention-based aggregation.

2.2 Graph Classification and Regression

Given the graph-level representation vector \(h_G\) for an input graph \(G\), there are usually two prediction tasks of interest:

1. classification, where a multi-layer perceptron (MLP) followed by the application of a non-linear activation function is used to map \(h_G\) to one of \(C\) predefined classes; and

2. regression, where an MLP maps \(h_G\) to a numeric value.

For a binary classification task (e.g., the MolHIV classification task) we normally use the sigmoid function and hence the final class label is obtained as follows:

\[\hat{l}^G = \text{Sigmoid}(\text{MLP}(h_G))\]  

In multi-class classification tasks (e.g., the MNIST digit recognition task [21]), we usually apply the SoftMax function, and the output class label is obtained as follows:

\[\hat{l}^G = \text{SoftMax}(\text{MLP}(h_G))\]  

For regression tasks (e.g., the PCQM4M regression task [28]), the output value is obtained directly from the MLP as follows:

\[\hat{l}^G = \text{MLP}(h_G)\].

3 METHODOLOGY

In this section we describe the technical details of our approach to graph classification and regression tasks. The different steps in our approach are schematically depicted in Figure 1.

In what follows, we fix an arbitrary set of training examples \(((G_1, l_1), \ldots, (G_n, l_n))\) consisting of pairs \((G_i, l_i)\) of a graph \(G_i\) and a corresponding prediction \(l_i\) (i.e., a class label in the case of classification, or a number in the case of regression). Furthermore, we fix a GNN given by a set of message-passing, update, and ReadOut functions as described in Section 2.

3.1 Phase 1: Initial Training and Indexing

**GNN Training** The first step in our approach is to train the GNN in a standard supervised way using the given examples.

As a result, we obtain a trained GNN model \(M\) with concrete values for each of the parameters of the model; these parameter values will remain fixed in all subsequent steps.

**Indexing of Training Examples** After training, we apply \(M\) to each of the example graphs \(G_1, \ldots, G_n\) and obtain the corresponding graph-level representation vectors \(h_{G_1}, \ldots, h_{G_n}\) as described in Equation (3). We then input key value pairs \(\{h_{G_i}, (G_i, l_i)\}\) for each
1 ≤ i ≤ n to the FAISS retrieval engine developed by Facebook [16] to construct an index from these key-value pairs. Using a FAISS index will significantly facilitate the retrieval of similar graphs at testing time. It is worth emphasizing that the index only needs to be constructed once throughout our entire pipeline.

3.2 Phase 2: Computing Predictions

Given an input graph \( G \) at testing time, we compute the corresponding prediction \( l \) according to the following steps.

First, we apply the trained GNN \( M \) to \( G \) to obtain, in the usual way, an initial prediction \( l_G \) and graph-level representation \( h_G \).

The main novelty in our approach, however, lies in the following two steps, where we first retrieve a set of most similar graphs from the index and subsequently exploit self-attention to compute the final prediction \( l \). We next describe these steps in detail.

Graph Retrieval. In this step, we use the FAISS index to find the the top-\( k \) vectors \( \{ h_G, \ldots, h_G \} \) that are most similar to \( h_G \), and to subsequently retrieve the corresponding example graphs and labels \( (G_i, l_i), \ldots, (G_i, l_i) \). The value \( k \) is considered a hyperparameter in our approach, and we use L2 distance to compute vector similarity scores.  

\[ d = \sqrt{\sum_{i=1}^{d} (v_i - \bar{v})^2} \]

Note that, since the index is built from the training dataset, the most similar graph for each input graph is always itself during the training stage. Inspired by [15] and to reduce overfitting, we also adopt the retrieval dropout strategy. Specifically, during the training stage, we will retrieve \( k + 1 \) graphs, in which the most similar graph will be dropped. In the evaluation stage, retrieval dropout is disabled, as evaluation graphs are usually not part of the training data.

Learnable Adapter Module and Self-Attention. Inspired by the transformer proposed in [36], we exploit self-attention to compute the final prediction \( l \) for input graph \( G \) as a weighted sum of the initially predicted label \( l_G \) and the top-\( k \) example labels \( l_1, \ldots, l_k \) with respective weights \( w_0, \ldots, w_k \).

The self-attention mapping takes as input a a query vector and a set of key-value pairs and yields the weights \( w_0, \ldots, w_k \). In our setting, the query vector \( q \) is obtained by multiplying a learnable matrix \( W_q \) to the graph-level representation \( h_G \) for the input \( G \):

\[ q = W_1 \cdot h_G, \]

In turn, to obtain the keys, we pack the \( k + 1 \) graph representations \( \{ h_G, h_G, \ldots, h_G \} \) as row vectors into a matrix \( H \), and then multiply \( H \) with a learnable matrix \( W_2 \) to obtain matrix

\[ K = W_2 \cdot H, \]

Finally, the attention weights \( Attn = (w_0, \ldots, w_k) \) are obtained computing the scaled product of \( q \) with the transpose of \( K \) and
Algorithm 1 Pipeline of Model Training

**Require:** training and validation datasets; randomly initialised GNN and adapter; training epochs \( m_1 \) for GNN and \( m_2 \) for the adapter; and number of retrieved graphs \( k \).

1. for \( i = 1 \) to \( m_1 \) do
2. \( \quad \) Train the GNN model over the training set and evaluate on the validation set in different epochs to choose the optimal GNN parameters to obtain a trained model \( M \)
3. end for
4. Construct FAISS index.
5. for \( i = 1 \) to \( m_2 \) do
6. \( \quad \) Train the Adapter over the training set using the index and the fixed model \( M \) and evaluate on the validation set in different epochs to choose the optimal parameters of the adapter.
7. end for
8. return GNN model \( M \) and Adapter defined by \( W_1, W_2, b \).

The overall training pipeline is illustrated in Algorithm 1. We set \( k \) by range search on the development set. We use cross entropy and mean square error as the loss functions for classification and regression tasks, respectively.

Lines 1–4 correspond to the initial training of the GNN model and the construction of the index, as described in Section 3.1. In Lines 5–7 we train the adapter module and optimise the values in the learnable matrices \( W_1 \) and \( W_2 \) and the bias vector \( b \) described in Section 3.2. At this point, the GNN model \( M \) is considered fixed.

Finally, when learning \( W_1 \) and \( W_2 \) from the training data using our entire pipeline, it is worth noticing that the most similar graph of an example graph \( G_i \) will be itself, so we will always drop it and consider only the other retrieved graphs.

4 EXPERIMENTS

In this section, we conduct comprehensive experiments on 12 datasets to answer the following questions:

**Q1:** Why do we need to do the graph retrieval?

**Q2:** Is only using the graph retrieval good enough?

**Q3:** Why do we need to use the self-attention?

**Q4:** How does the quality of retrieved results affect the performance of GraphRetrieval?

**Q5:** How do different base models affect the performance of GraphRetrieval?

**Q6:** How does GraphRetrieval perform when compared with the base models?

We present the results of an extensive evaluation on graph classification and regression tasks. In Section §4.1, we introduce the benchmark datasets. In Section 4.2, we describe the baseline GNN models. In Section 4.3, we provide additional details on the training process. In Section 4.4, we analyse the influence of graph retrieval quality on overall performance and investigate the use of different pre-trained models for graph retrieval. Our main results are described in Section 4.5. In Section 4.6, we conduct an ablation study to determine the influence of different elements of our approach. In Section 4.7, we explore an alternative approach for training and finally, in Section 4.8, we provide a more detailed case-study using the best performing GNN baseline.

### 4.1 Benchmark Dataset

We used a collection of 12 datasets consisting of 8 small-scale molecule datasets, 2 computer vision datasets, and 2 large-scale quantum chemistry datasets. The statistics of each dataset are summarized in Table 1. The molecule datasets and the computer vision datasets focus on classification tasks, while PCQM4M and PCQM4Mv2 datasets are relevant for regression tasks. For MNIST and CIFAR10, we pre-process the images into grids before applying GNNs. For a fair comparison, we follow the standard train/valid/test splits for all datasets according to ogb\(^2\) and torch\(_{\text{geom}}\).\(^3\) In particular, for the molecule datasets, we use scaffold split [29], where molecules are split according to their scaffold (molecular substructure). We used a conventional random split for the remaining datasets.

\(^2\)https://github.com/snap-stanford/ogb

\(^3\)https://github.com/pyg-team/pytorch\(_{\text{geom}}\)
We report the best result on the development sets. We adopt three widely-used GNN baselines: Graph Convolutional Networks (GCN) [19], a classic type of GNN that can model graph-structured data; Graph Isomorphism Networks (GIN) [39], which is theoretically more expressive compared to GCN and shown to be as powerful as the Weisfeiler Lehman graph isomorphism test; and Principal Neighbourhood Aggregation (PNA) [5], which combines multiple aggregators and logarithmic degree scalers to better generalise the injectivity property and distinguish their neighbourhoods.

Table 1: Statistics of datasets and their corresponding evaluation metrics used in our experiments. MCC denotes multi-class classification, MLC denotes multi-label classification, and BC denotes binary classification. For PCQM4M and PCQM4Mv2, their testing datasets are not publicly available. We report the best result on the development sets.

| Dataset     | Train   | Valid  | Test    | Task         | Metric    |
|-------------|---------|--------|---------|--------------|-----------|
| MNIST       | 55000   | 5000   | 10000   | MCC          | Accuracy  |
| CIFAR10     | 45000   | 5000   | 10000   | MCC          | Accuracy  |
| BBBP        | 1631    | 204    | 294     | BC           | ROC_AUC   |
| Tox21       | 6264    | 783    | 784     | 12-label MLC | ROC_AUC   |
| ToxCast     | 6860    | 858    | 858     | 617-label MLC | ROC_AUC   |
| SIDER       | 1141    | 143    | 143     | 27-label MLC | ROC_AUC   |
| ClinTox     | 1181    | 148    | 148     | 2-label MLC  | ROC_AUC   |
| MUV         | 74469   | 9309   | 9309    | 17-label MLC | ROC_AUC   |
| HIV         | 32901   | 4113   | 4113    | BC           | ROC_AUC   |
| BACE        | 1210    | 151    | 152     | BC           | ROC_AUC   |
| PCQM4M      | 3,045,360 | 380,670 |        | regression   | MAE       |
| PCQM4Mv2    | 3,378,606 | 73,545  |        | regression   | MAE       |

Table 4.3 Training Details

4.2 Baseline Models

We adopt three widely-used GNN baselines: Graph Convolutional Networks (GCN) [19], a classic type of GNN that can model graph-structured data; Graph Isomorphism Networks (GIN) [39], which is theoretically more expressive compared to GCN and shown to be as powerful as the Weisfeiler Lehman graph isomorphism test; and Principal Neighbourhood Aggregation (PNA) [5], which combines multiple aggregators and logarithmic degree scalers to better generalise the injectivity property and distinguish their neighbourhoods.

Besides, we adopt two additional approaches, denoted as Retrieval-only, which retrieves one most similar graph from the FAISS index; and Majority-voting, which retrieves a certain number of most similar graphs from the FAISS index and then chooses the most frequent label among these retrieved results as our predicted answer.

4.3 Training Details

We have implemented our approach using Pytorch v. 1.7.0 in a system called GraphRetrieval. Our system is model-agnostic and can be applied to enhance a wide range of GNN baselines. Training is conducted as described in Algorithm 1. For the first phase of the training (Lines 1–4 in Algorithm 1), we use the Adam optimizer [18] with an initial learning rate 0.01 and decay the learning rate by 0.5 every 50 epochs for all the GNN baselines. Batch normalization [14] is applied on every hidden layer. Some hyperparameters for the three baseline GNN models are shown in Phase 2 of the training (Lines 5–7) is repeated 5 times with different initialisation seeds for W₁, W₂, and b; we report the mean and standard deviation of the values obtained in these runs.

4.4 Graph Retrieval

GNN models without the graph retrieval (Q1). Despite their success in application, GNN models are also known to suffer an important practical limitation – that is, example forgetting [34]. To empirically demonstrate the example forgetting problem, we design an experiment whose setting is: given a certain number of training examples our trained model has already seen, we want to see how our trained model performs on these seen examples.

Table 3: Results on 1000 training (seen) examples of CIFAR10 and MNIST. The evaluation metric is accuracy.

|                  | CIFAR10 | MNIST |
|------------------|---------|-------|
| GCN              | 0.901   | 0.982 |
| GIN              | 0.910   | 0.984 |
| PNA              | 0.927   | 0.986 |
| Retrieval-only   | 1       | 1     |

From Table 3, we can observe that for the 1000 training examples, three GNN models are unable to correctly predict all ground-truth labels. In particular, for the samples from the CIFAR10 dataset, 99, 88, 73 out of the 1000 training samples are incorrectly predicted by GCN, GIN and PNA, respectively. However, for the results of Retrieval-only, it is as expected that we obtain 100% accuracy on the seen examples. It is reasonable that a well-trained GNN model is not expected to fit its parameters to all training examples for a better generalisation performance. However, for some applications, it might be not acceptable for a model incapable of predicting seen samples completely and correctly. Therefore, if we can make use of the retrieved results, it will help a well-trained model correct some mispredicted results, thereby improving the model performance.

Predict with only graph retrieval (Q2). Here, we explore whether only using the retrieved results is good enough to achieve a good prediction. We compare Retrieval-only and Majority-voting methods against three well trained models on four datasets: CIFAR10, MNIST, PCQM4M and PCQM4Mv2. In particular, for Majority-voting method, we try three different retrieved numbers (5, 10 and 20) to see their performance on four different datasets.

From Table 4, we can clearly see that both Retrieval-only method and Majority-voting method greatly underperform the three well-trained models. In the CIFAR10 dataset, Retrieval-only methods have more than 15% accuracy loss when compared with the other three well-trained GNN models. For Majority-voting method, we can notice that as we increase the number of retrieved graphs from 10 to 20, the accuracy decreases on both CIFAR10 and MNIST datasets. One straightforward reason might be that as the number of retrieved results increases, the number of noisy labels will increase accordingly. For PCQM4M and PCQM4Mv2 datasets, we can observe that Majority-voting method performs the same regardless of the changes of the number of retrieved results. Recall
that Majority-voting method takes the most frequent label among the retrieved results as its predicted answer, so for PCQM4M and PCQM4Mv2, two datasets used for the regression task, their label range is continuous; we notice that the each of retrieved results from PCQM4M and PCQM4Mv2 datasets has a different label and the label frequency is always 1. Since we take the label of the most similar graph as our predicted answer, Majority-voting method thus degrades to behave the same as Retrieval-only method. This explains why using different retrieved numbers in Majority-voting method, we obtain the same MAE scores on both PCQM4M and PCQM4Mv2 datasets. Of course, for the regression task, another workable solution is to use the average value of the retrieved results as the predicted answer, and we will report the results of such an average strategy in the subsequent paragraph.

Self-attention VS Averaging (Q3). One intuition about using the self-attention is that we hope to assign different weights to different retrieved results according to their relevance with respect to the target graph. Here, to demonstrate the necessity and the effectiveness of using the self-attention mechanism in our framework, we compare a self-attention setting against an averaging setting. Assume we have a predicted label \( l_G \) from the trained GNN \( M \) and \( k \) labels \( \{l_1, \ldots, l_k\} \) from the retrieved results, we need to output a predicted answer. The the self-attention mechanism refers to the phase 2 described in Section 3.2, in which we assign different weights to the \( k+1 \) labels \( \{l_1, \ldots, l_k, l_G\} \) and then get the predicted answer based the weighted sum as shown in Equation 10. For the averaging setting, we different take the average value of \( k+1 \) labels as our predicted without distinguishing their degree of importance to our prediction. We experiment with the two different strategies on two classification datasets (CIFAR10 and MNIST) and two regression datasets (PCQM4M and PCQM4Mv2). In particular, for the classification task in which each label is an integer, we thus omit the decimal fractions smaller than 0.5 and counting all others, including 0.5, as 1, i.e., an average value 6.3 is considered as 6 while 6.7 is considered as 7. Table 5 illustrates the experimental results.

From Table 5, we can observe that incorporating the self-attention mechanism into our framework significantly outperforms than using the averaging scheme. One possible reason is that constrained by the performance of the retrieval systems, the model should not assign the same weight to the retrieved labels and the label obtained by the well-trained model in computing the final prediction. Therefore, adopting a self-attention to model the relevance between the target graph and the retrieved graphs is considered to be an effective way to make use of the retrieved results (when needed) while avoid sabotaging the performance of the well-trained model.

**Quality of the retrieval (Q4).** We use the following metrics to assess the quality of the retrieved graphs: average retrieval accuracy (ARA) for classification tasks and average retrieval MAE (ARM) for regression tasks. The ARA metric quantifies the proportion of retrieved labels that are identical to the ground-truth label. In turn, the ARM metric relies on the average retrieval MAE to quantify the differences between the retrieved labels and the ground-truth label. Specifically, let us assume that we have \( n \) graphs with their corresponding ground truth labels \( \{l_1, \ldots, l_n\} \), and we retrieve the top-\( k \) most similar graphs for each of them, with labels \( \{l_1^i, \ldots, l_k^i\} \) for each \( 1 \leq i \leq n \). Then, the ARA metric is defined by the following expression, where \( 1() \) denotes the indicator function:

\[
ARA = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{k} \sum_{j=1}^{k} 1(l_j^i = l_i).
\]

In turn, the ARM metric is given by the following expression:

\[
ARM = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{k} \sum_{j=1}^{k} |l_j^i - l_i|.
\]

Figures 4, 5 and 6 plot the values of the ARA and ARM metrics for different numbers of retrieved graphs. Since the molecule datasets are highly imbalanced, we not only report the ARA for all samples, but also report the metric on the positive samples, as we believe the retrieval quality for positive samples will directly affect the final performance. It can be observed that the number of retrieved graphs directly affects these two metrics. Figure 4 shows that retrieving a single graph achieves the best performance on ARA for the molecule datasets, which suggests that retrieving a large number of graphs introduces noise in such imbalanced datasets. In the case of CIFAR10, MNIST, PCQM4M and PCQM4Mv2 we obtain the best performance for \( k = 3 \) for both ARA and ARM; retrieving a larger number of graphs does not improve performance in any of these cases. These findings are consistent with the results in Tables 8, 9, and 10, which demonstrates that the ARA and ARM metrics are highly correlated with the final performance of the model, and hence are well-suited for measuring retrieval quality.

**Using Different GNNs for Graph Retrieval and Adapter (Q5).** We study the possibility of using different GNN models \( M_1 \) and \( M_2 \) at different stages of our approach. In particular, we trained

| Method         | CIFAR10 | MNIST | PCQM4M | PCQM4Mv2 |
|---------------|---------|-------|--------|---------|
| Self-attention| 0.676   | 0.972 | 0.160  | 0.120   |
| Averaging     | 0.577   | 0.921 | 0.313  | 0.214   |

Table 5: Experimental results of Self-attention VS Averaging on four datasets (k=3). The base model we use is GCN.
models $M_1$ and $M_2$ independently using the training and validation datasets (c.f. Lines 1–3 in Algorithm 1). We use $M_1$ to build the FAISS index (c.f. Line 4) and also to retrieve the most similar graphs to the input graph at testing time (see Section 3.2). In turn, we used $M_2$ to train and apply the self-attention adapter module.

We set $M_2$ to be a GCN, and we repeated the experiments on the CIFAR10 dataset with three different choices for $M_1$, namely GCN, GIN and PNA, as the graph retrieval module on CIFAR10. As shown in Table 6, we obtained the best performance by choosing PNA as the model for graph retrieval. This is consistent with our previous results in Table 9, and further confirms that the quality of retrieval directly affects overall performance.

**Random vs. Similarity-based Retrieval** We next investigate the effects of noisy retrieved graphs on the overall performance of our approach. In particular, we studied the situation where graphs in the training set are retrieved at random, rather than based on their similarity to the input graph. We hypothesise that, with random retrieval, the model will assign lower self-attention weights to the retrieved graphs and a much higher weight to the input graph. This shows that our model is noise-tolerant and will naturally assign weights to the retrieved graphs based on their usefulness.

### 4.5 Main Results (Q6)

Following previous studies [12, 13], we adopted ROC-AUC as the evaluation metric for the molecule datasets; in turn, we used accuracy to evaluate classification performance for the MNIST and CIFAR10 datasets, and we used Mean Average Error (MAE) for regression tasks in PCQM4M and PCQM4Mv2. The obtained results are summarised in Tables 8, 9 and 10 respectively.
ARA for both negative and positive samples in HIV

Table 8: Experimental results on 8 molecule datasets (k=1).

|        | BBBP  | Tox21 | ToxCast | SIDER | ClinTox | MUV    | HIV    | BACE | Average |
|--------|-------|-------|---------|-------|---------|--------|--------|------|---------|
| GCN    | 70.0±0.3| 73.9±1.8| 64.0±2.3| 58.6±2.1| 92.2±2.7| 75.9±2.0| 78.6±0.3| 81.9±0.9| 74.4    |
| GIN    | 70.5±0.7| 74.9±0.2| 64.5±1.0| 58.7±2.1| 87.0±3.0| 76.8±1.8| 77.0±0.2| 74.8±0.7| 73.0    |
| PNA    | 70.0±1.9| 73.0±0.3| 62.0±1.9| 58.0±0.2| 87.0±2.3| 71.0±3.5| 77.0±0.6| 72.0±0.5| 71.2    |

Table 9: Experimental results on CIFAR10 and MNIST (k=3).

|        | CIFAR10 | MNIST |
|--------|---------|-------|
| GCN    | 0.647   | 0.960 |
| GIN    | 0.656   | 0.968 |
| PNA    | 0.687   | 0.972 |

GraphRetrieval-GCN 0.676(2.9%)† 0.972(1.2%)†
GraphRetrieval-GIN 0.682(2.4%)† 0.979(1.1%)†
GraphRetrieval-PNA 0.723(3.6%)† 0.982(1.0%)†

Table 10: Performance on PCQM4M and PCQM4Mv2 (k=3).

|        | PCQM4M | PCQM4Mv2 |
|--------|--------|----------|
| GCN    | 0.171  | 0.139    |
| GIN    | 0.157  | 0.123    |

GraphRetrieval-GCN 0.160(0.011)† 0.120(0.019)†
GraphRetrieval-GIN 0.142(0.015)† 0.112(0.011)†

Our empirical results have demonstrated the efficacy of our approach on different datasets. In what follows, we conduct additional experiments with the goal of gaining a better understanding of the different factors influencing the efficacy of our approach.

Table 11: Ablation study of retrieval dropout and ranking encoding. Similar results are obtained on the other datasets.

|        | CIFAR10 | MNIST      |
|--------|---------|------------|
| GCN    | 0.647   | 0.960      |
| GIN    | 0.656   | 0.968      |
| PNA    | 0.687   | 0.972      |

4.6 Ablation Study

We studied the influence of retrieval dropout (c.f. Section 3.2) and the use of a learnable bias vector in Equation (9). Our results are summarised in Table 11. We can conclude from these results that dispensing with retrieval dropout and the bias vector has a clear negative effect on performance.

4.7 Alternative Training Pipelines

As already mentioned in our discussion of Algorithm 1, we train our approach in two stages: first, we train the GNN model M and then we fix M and train the adapter by optimising the self-attention parameters. An alternative approach would be to use the adapter training stage to further optimise the parameters of the GNN model M. The latter approach will lead to longer training times, as the number of free parameters has increased, but it may also result in better overall performance. We next compare the performance...
of these two approaches on the CIFAR10 dataset. Our results are summarised in Figure 7. The upper half of the figure plots the average run time per epoch; as expected, training times are doubled if we adopt the alternative approach. The lower half of the figure depicts overall performance; surprisingly, our approach achieves better performance than the alternative consistently across all three baseline GNN models. This suggests that keeping the parameters of the GNN model M fixed while training the adapter simplifies the training process and makes it more stable as a result.

4.8 Case Study Using PNA as GNN Baseline

To conclude this section, we provide a more detailed analysis of the performance of our approach when using PNA as a GNN baseline. For this, we randomly selected 1,000 graph samples from the testing dataset of CIFAR10 to compare the prediction results of PNA and our GraphRetrieval approach using also PNA as pre-trained GNN model. The results are summarised in Table 7. We identified 636 cases where both the baseline and our approach predicted the label correctly, and 170 cases where both predicted the label incorrectly. In turn, while we found 65 cases where the baseline correctly predicted the label but our approach obtained the wrong prediction, we also identified 129 examples where the baseline reported the wrong prediction but our approach produced the correct one. These results suggest that demonstrates that the benefits brought by graph retrieval outweigh its disadvantages.

5 RELATED WORK

In this section, we briefly discuss recent GNN architectures and retrieval-augmented models in Deep Learning.

5.1 Graph Neural Networks

Graph Neural Networks have become the de-facto standard for Machine Learning tasks over graph-structured data. The fundamental idea behind GNNs is that of Message Passing—that is, to iteratively update each node-level representation by aggregating information from its neighborhoods throughout a fixed number of layers [2, 8, 10, 19, 39].

Architecture design is typically the main factor for introducing inductive biases. In molecule classification applications, Weave [17] explicitly learns both the atom- and bond-level representation during message passing; D-MPNN [40] emphasizes the information delivered along different directions; more recently, N-Gram Graph [26] and AWARE [6] focus on modeling the walk-level information on molecules.

The choice of the aggregation function is another key aspect of GNN design. GAT [37] adds an attention module; GGNN [23] combines information from its neighbors and its own representation with a GRU unit [4]; finally, PNA [5] adopts multiple aggregation functions for combining messages.

5.2 Retrieval-augmented Models

Prior studies in different domains show that models depending only on input features and learned parameters are less powerful than models augmented by external knowledge [11, 20, 22, 25, 38].

The authors of [38] propose the use of a kNN-augmented attention layer for language modeling tasks. In [20], the authors advocate for the use of the entire dataset (rather than using just a single datapoint) for making predictions; in particular, they show that there exist meaningful dependencies between the input and the training dataset which are lost during training. Other related works such as REALM [11] and KSTER [15] augment the Transformer model by retrieving relevant contexts in a non-parametric way.

In contrast to prior work, however, GraphRetrieval focuses on the retrieval of graph-structured data. Retrieving similar graphs can be much more challenging than retrieving similar text, where pre-trained models like BERT [7] can provide universal representations for measuring similarities. Such pre-trained models are missing for diverse graph-structured data. Furthermore, graph retrieval is also negatively affected by over-smoothing and over-squashing in existing GNNs.

6 CONCLUSION AND FUTURE WORK

We have proposed GraphRetrieval, a model-agnostic approach for improving the performance of GNN baselines on graph classification and regression tasks. Our main contribution is a method for exploiting the retrieval of similar graphs in the training set for making predictions, and a learnable adapter module based on self-attention for determining the influence of each retrieved graph on the final prediction. Our empirical results clearly show that GraphRetrieval consistently improves the performance of existing GNN architectures on a very diverse range of benchmarks. We believe that our results open the door for the further development of sophisticated methods for explicitly exploiting the training data when making predictions on graphs.

Our research opens a number of interesting avenues for future work. First, it would be interesting to extend our approach to tackle node-level classification tasks. Furthermore, our approach is rather general and flexible, and we believe that it can be applied to advanced methods based on 3D geometric modeling [27, 30, 32, 33] as well as to a broad range of additional tasks [10, 35].

REFERENCES

[1] Alon, U., and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In ICLR (2021).
[2] Balclutha, M., Heroux, P., Gauzere, B., Vasquez, P., Adam, S., and Honeine, P. Breaking the limits of message passing graph neural networks. In ICLR (2021).

[3] Cai, C., and Wang, Y. A note on over-smoothing for graph neural networks. In CoRR (2020).

[4] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. In arXiv preprint arXiv:1409.1255 (2014).

[5] Corso, G., Cavallari, L., Brani, D., Lio, P., and Veličković, P. Principal neighbourhood aggregation for nets. In NeurIPS (2020).

[6] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. NeurIPS (2015).

[7] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In ICML (2017).

[8] Gurev, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M. E. Neural message passing for protein structure prediction. In ICLR (2020).

[9] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Embedding proteins into hyperbolic space for predicting their functions. In NeurIPS (2018).

[10] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Catasta, M., and Leskovec, J. Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[11] Jia, J., and Tenenbaum, J. B. Deep networks for structured data. arXiv preprint arXiv:1708.01601 (2017).

[12] Johnson, J., Douze, M., and Jegou, H. Billion-scale similarity search with gpus. IEEE Transactions on Big Data (2019).

[13] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[14] Kipf, T. N., and Welling, M. Deeper understanding of graph convolutional networks. In ICLR (2017).

[15] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[16] Kohler, M., and Leskovec, J. Learning to rank with graph convolutional networks. In CoRR (2020).

[17] Konstantinou, D., and Pan, H. A survey on graph neural networks. arXiv preprint arXiv:2003.08032 (2020).

[18] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[19] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[20] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[21] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In ICLR (2017).

[22] Lewis, M., Ghazvininejad, M., Ghosh, G., Agnihotry, A., Wang, S., and Zettlemoyer, L. Pre-training via paraphrasing. In NeurIPS (2020).

[23] Li, Y., Tallow, B., Brockschmidt, M., and Zemel, R. Rethinking graph neural networks. In arXiv preprint arXiv:1711.05495 (2015).

[24] Liu, Q., Alam, S., Brockschmidt, M., and Gaunt, A. L. Constrained graph variational autoencoders for molecule design. In NeurIPS (2018).

[25] Liu, Y., Yogatama, D., and Blunsom, P. Relational memory augmented language models. arXiv preprint arXiv:2201.09680 (2022).

[26] Liu, S., Demmel, M. F., and Liang, Y. N-Gram graph: Simple unsupervised representation for graphs, with applications to molecules. CoRR (2019).

[27] Liu, Y., Wang, L., Liu, M., Zhang, X., Ozerkar, B., and Ji, S. Spherical message passing for 3d graph networks. In ICLR (2022).

[28] Nakaya, M., and Shimaoka, T. Pulchene: A large-scale first-principles electronic structure database for data-driven chemistry. In Journal of chemical information and modeling (2017).

[29] Ramsundar, B., Eastman, P., Walters, P., and Pan, D. Deep learning for the life sciences: applying deep learning to genomics, microscopy, drug discovery, and more. O'Reilly Media, 2019.

[30] Satorras, V. G., Hoogenboom, E., and Welling, M. E. (n) equivariant graph neural networks. In ICML (2021).

[31] Schlichtkrull, M. S., Kepp, T. N., Breshears, D., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In CoRR (2017).

[32] Schütt, K., Kindermans, P.-J., Sauceda Felix, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In NeurIPS (2017).

[33] Schütt, K., Urne, O., and Gasteiger, M. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In ICML (2021).

[34] Schütt, K., Gasteiger, M., and Gasteiger, M. An empirical study of example forgetting during deep neural network learning. In NeurIPS (2019).

[35] Townshend, R. J., Vögele, M., Suriana, P., Derry, A., Powers, A., Laloudakis, T., and Derry, A. Atom3d: Tasks on molecules in three dimensions. In NeurIPS (2021).

[36] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In NeurIPS (2017).

[37] Veličković, P., Cucurull, G., Casanova, A., Romero, X., Bengio, Y., and Gori, M. Graph attention networks. In arXiv preprint arXiv:1710.10903 (2017).

[38] Wu, Y., Bane, M. N., Hutchins, D., and Sendik, N. Memorizing transformers. In ICLR (2022).

[39] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How well are graph neural networks? In ICLR (2018).

[40] Yang, K., Swanson, K., Jin, W., Coley, C., Eiden, P., Gao, H., Guzman-Perez, A., Hopper, T., Kelley, B., Mathea, M., et al. Analyzing learned molecular representations for property prediction. In Journal of chemical information and modeling (2019).

[41] Ying, R., He, R., Chen, K., Yatskar, M., and Leskovec, J. Graph convolutional networks for web-scale recommender systems. In KDD (2018).

[42] Yogatama, D., de Masson Bautume, C., and Kong, L. Adaptive semiparametric language models. TACL 9 (2021), 362–373.