ABSTRACT

Title of Dissertation: TOWARDS GENERALIZED AND SCALABLE MACHINE LEARNING ON STRUCTURED DATA

Kezhi Kong
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

Deep Learning and Neural Networks have brought a transformative era for the field of machine learning, significantly influencing how we approach and utilize structured data. This dissertation is dedicated to exploring machine learning methodologies specifically designed for structured graphs and tables, aiming to enhance the performance of neural networks on the important data modalities.

Graph Neural Networks (GNNs) have emerged as powerful architectures for learning and analyzing graph representations. However, the training of GNNs on large-scale datasets usually suffers from overfitting, posing significant generalization challenges for prediction problems. Meanwhile, conventional GNNs are hindered by scalability problem when deployed on industrial-level graph datasets. Moreover, for the table reasoning task, Large Language Models (LLMs) have shown competitive ability but cannot fully process large tables due to context limit and may fail to comprehend the complex relationships within tabular data. In this dissertation, we investigate

algorithms and techniques to address the generalization and scalability issues of GNNs, as well as the effective and efficient approach to the table reasoning task.

In the first work, we propose to leverage data augmentation to generalize GNNs. We propose FLAG (Free Large-scale Adversarial Augmentation on Graphs), which iteratively augments node features with gradient-based adversarial perturbations during training. In the second and third work, we investigate GNNs' scalability problem. We propose VQ-GNN, a universal framework to scale up any convolution-based GNNs using Vector Quantization (VQ) without compromising the performance. We further propose GOAT, a global graph transformer that scales to large graphs with millions of nodes and is competitive on tasks of both homophilious and heterophilious graphs. Lastly, we propose OpenTab, an effective method towards open-domain table reasoning task built with the advanced Large Language Models.

TOWARDS GENERALIZED AND SCALABLE MACHINE LEARNING ON STRUCTURED DATA

by

Kezhi Kong

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Tom Goldstein, Chair/Advisor
Professor Joseph F. JaJa
Professor Furong Huang
Professor Sanghamitra Dutta
Professor Tianyi Zhou

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me throughout the PhD journey.

First and foremost, I would like to express my profound gratitude to my supervisor, Prof. Tom Goldstein. Grinding as a PhD has been challenging, yet your insights, expertise, guidance, and encouragement lightened the journey. It has been an immense privilege to work under the guidance of someone as knowledgeable and approachable as you. You are a smart and cool professor and will always be my role model.

I am also deeply grateful to my committee members, Prof. Joseph F. JaJa, Prof. Furong Huang, Prof. Sanghamitra Dutta, and Prof. Tianyi Zhou. Your willingness to serve on my committee and the invaluable feedback you provided on my research and dissertation have been instrumental in my academic development. Additionally, I must extend my thanks to Prof. John Dickerson and Prof. Ming Lin for their instructions and support.

I also want to thank the mentors whom I met during the internships, who are Chen Zhu, Jiani Zhang, Mohammad Shoeybi, Parsa Saadatpanah, and Zak Jost. Your invaluable instructions and guidance have been profoundly impactful. Thank you for your dedication and for playing such a pivotal role in my professional development.

I would like to express my heartfelt appreciation to my friends and collaborators for their support and assistance. To Qihang Li, Songwei Ge, Haoan Feng, Renkun Ni, Manli Shu, Ping-yeh Chiang, John Kirchenbauer, Jonas Geiping, Chen Chen, Guohao Li, Jiuhai Chen, Mucong Ding, Jingling Li, and many others, your collaboration and friendship have been invaluable to me. The bond we share and the collective effort we have put into our work mean a lot.

I must extend gratitude to my parents for their unwavering belief in me and their unconditional support. I am so grateful you have always been respecting my decisions and supporting me to chase my dream. Your trust and love mean everything to me.

Lastly I thank my girlfriend Baobao Liu, who has always been my biggest supporter and cheerleader throughout the PhD journey. Your presence turns tough days into bearable ones, and joyful days into unforgettable memories. Your encouragement and love have made me brave and confident. I am grateful for your love and companionship, and all the happiness we share together.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In recent years, Neural Networks (NNs) have emerged as a powerful foundation model in the realm of machine learning, fundamentally altering our approach to understanding and interpreting complex data patterns. The NNs have brought significant advances in a multitude of domains, particularly in modeling structured data forms such as graphs and tables, which are important to representing complex relationships and interactions in scientific and practical applications [3, 4, 5, 6].

Structured data, with its inherent complexity and the rich information, presents a unique set of challenges and opportunities for machine learning. Graphs, representing entities and their interconnections, are essential for understanding complex networks such as social networks, biological networks, and transportation systems. Tables, on the other hand, are fundamental in organizing and analyzing data across various fields, from finance and healthcare to natural sciences and beyond. This dissertation delves into the cutting-edge methodologies and innovations aimed at enhancing the efficacy and efficiency of machine learning models when applied to structured graph and table data.

Graph Neural Networks (GNNs) have become a popular approach for learning and analyzing graph structures. One representative model is the Graph Convolutional Network (GCN) [1], and its various adaptations have been applied to several tasks like visual recognition [7], meta-learning [8], social analysis [9, 10], and recommender systems [11]. However, the training of GNNs on large datasets often leads to overfitting issues, and realistic graph datasets usually involve a significant number of test nodes that are out of the distribution [12], which presents challenges for prediction tasks.

To address overfitting in deep neural networks, one effective technique is data augmentation [13], which is commonly used in computer vision applications. Data augmentation involves applying label-preserving transformations to the inputs, such as translations and reflections for images. By doing this, data augmentation can effectively increase the size of the training set without adding significant computational costs. However, it is still an ongoing challenge to extend the concept of data augmentation to GNNs. In the case of image data, transformations rely heavily on the underlying image structure [14]. However, for non-visual tasks like natural language processing [15] and graph learning, it is difficult to design low-cost transformations that preserve semantic meaning. Generally, graph data for machine learning consists of both graph structure (or edge features) and node features. In the few cases where data augmentation is feasible for graphs, it usually focuses only on modifying the graph structure by adding or removing edges [16, 17, 18, 19, 20, 21]. In Chapter 2, we propose a data augmentation method that is based on adversarial training. The method is named as FLAG. FLAG is simple but effective. It adds adversarial data augmentation onto the input node features, which is highly scalable and easy to implement [22].

Moreover, GNNs face scalability issues when modeling large realistic graph datasets. As GNNs aggregate information from all L-hop neighbors in an L-layer structure, and standard training routines require all these neighbors to be materialized the GPU device in the mean time. This makes full-batch training infeasible for graphs with billions of millions of nodes. To overcome this issue, sampling-based methods have been proposed, which can be categorized into three groups: neighbor-sampling, layer-sampling, and subgraph-sampling methods. While these methods can speed up GNN training, they suffer from three main drawbacks. First, during inference, sampling methods require all neighbors to make non-stochastic predictions, which can be computationally expensive if the full graph cannot fit on the inference device. Second, state-of-the-art sampling-based methods fail to achieve satisfactory results across various tasks and datasets, as reported in [12] and in Section 3.6. Third, these methods are not universally applicable to GNNs that rely on many-hop or global context in each layer, which limits the application of more powerful GNNs to large graphs. Given these, in Chapter 3 we propse VQ-GNN, a universal scheme to scale up GNNs to large-scale graph applications. We explore the idea of using vector quantization (VQ) as a means of dimensionality reduction to scale up the state-of-the-art GNN models with reduced complexity from quadratic to linear. In Chapter 4 we further propose GOAT, a global graph transformer that can conceptually make each node attend to every other node in the graph. GOAT consists of both local and global convolutional modules that empower it to register competitive performances on large-scale homophilious and heterophious graph data [23, 24].

Many retrieval-based LLMs primarily target textual data from the web or text corpora, overlooking the wealth of information stored in structured tables. For the structured table data, Large Language Models (LLMs) outperform traditional methods and register as the new powerful state-of-the-art. When applied to the tables, retrieval-based LLMs encounter the following challenges: (1) Structured tables have diverse data types, which results in a high token usage and computational constraints. (2) LLMs may fail to comprehend the complex relationships within tabular data. (3) The restricted maximum context length of LLMs. In Chapter 5, we propose OPENTAB to address the table reasoning task based on LLMs. To mitigate challenges of LLMs in processing and understanding structured tables, we utilize an LLM as a CODER to generate high-quality SQL programs for efficient table parsing. Moreover, an LLM-based READER works to formulate the final response based on the SQL execution results. By breaking down the complex reasoning job into programmatic steps, we achieve enhanced accuracy and reliability in open-domain table reasoning [25, 26].

1.1 Preliminaries

Below we introduce Graph Neural Networks (GNNs), Large Language Models (LLMs), and the corresponding preliminaries.

1.1.1 Graph Neural Networks

Graph Neural Networks (GNNs) are important tools in learning and understanding complex and interacting systems that are abstracted as graphs. ChebNet [27] leveraged the spectral representation of the graphs using the K-localized convolution to avoid the computation of the eigenvectors of Laplacian matrices. Graph Convolutional Network (GCN) [1] mitigated the problem of overfitting the local neighborhood of graph structures and used the normalized adjacency matrices to realize convolution operations and message passing within the local neighborhood. Hamilton et al. [17] proposed a general inductive graph learning framework, GraphSAGE, which randomly samples and aggregates features from one node's neighborhood with three diverse kinds of aggregators which are Mean, LSTM, and Pooling. Graph Attention Network (GAT) [28] used the attention mechanism to compute the weighted average features from the neighbors. The network can employ multi-head attention schemes to enlarge the model capacity and is powerful on a number of transductive and inductive node classification benchmarks. Xu et al. [29] proposed a graph classifier that is theoretically as strong as the classic Weisfeiler-Lehman (WL) graph isomorphism test. The proposed Graph Isomorphism Network (GIN) used the summation computation as the aggregator and outperformed other baselines on a range of graph classification benchmarks.

We denote a graph as $G(V, E)$. For initial node features, we use x_v for $v \in V$, and e_{uv} for $(u, v) \in E$ for edge features respectively. The learned representation vectors are denoted as h_v for every node $v \in V$. For the entire graph G, the graph feature vector h_G is learned to downstream graph property prediction tasks. We represent GNNs using the message-passing scheme, where the k-th iteration of message passing, i.e. the k-th layer of GNN forward computation is as below:

$$h_{\mathcal{N}(v)} = \text{AGGREGATE}^{(k)} \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right), \tag{1.1}$$

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, h_{\mathcal{N}(v)} \right), \tag{1.2}$$

where $h_v^{(k)}$ is the feature embedding of node v after k iterations of message passing, e_{uv} is the edge feature vector, $\mathcal{N}(v)$ is the neighbor set of node v, $h_{\mathcal{N}(v)}$ is the aggregated feature from the neighborhood, and $h_v^{(0)} = x_v$. COMBINE$(\cdot)$ and AGGREGATE$(\cdot)$ are functions parameterized by feed-forward neural networks. To simplify, we view the whole pipeline as an end-to-end function $f_\theta(\cdot)$ built on graph $\mathcal{G}$:

$$H^{(K)} = f_\theta(X; \mathcal{G}), \tag{1.3}$$

where X is the input node embedding matrix. After K rounds of graph convolution we get the final node matrix $H^{(K)}$. For the representation embedding of the entire graph h_G, the permutation-invariant READOUT$(\cdot)$ function pools node features from the final iteration K as:

$$h_G = \text{READOUT}\left(\left\{h_v^{(K)} \mid v \in V\right\}\right), \tag{1.4}$$

Above are GNNs summarized from the point of view of message passing. Additionally the spectral convolution viewpoint is also widely adopted in favor of the theoretical analysis of GNNS, and the k-th layer of GCN can be shown as:

$$I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \tag{1.5}$$

$$S = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \tag{1.6}$$

$$H^{(k+1)} = \sigma\left(S H^{(k)} \theta^{(k)}\right), \tag{1.7}$$

where σ is the activation function, $H^{(k)}$ is the node feature matrix of the k-th layer with $H^{(0)} = X$, and θ^k is the trainable weight matrix at the layer k. D and A denote the diagonal degree matrix and adjacency matrix, respectively. Here, we treat S as a normalized adjacency matrix with self-loops added.

1.1.2 Scaling up Graph Neural Networks

Here, we introduce several recent scalable algorithms for Graph Neural Networks (GNNs) and analyzing their memory and time complexities. The focus is on algorithms that can be applied to agnostic GNN models, including NS-SAGE [17], Cluster-GCN[30] , and GraphSAINT[31].

We use GCN as a simple benchmark, and when applied to a sparse graph with n nodes and m edges, we denote the number of GNN layers as L, b as the mini-batch size, f as the hidden feature dimension, and d as the average node degree. We categorize the sampling baselines into three groups.

Neighbor-sampling. The neighbor-sampling scheme selects a subset of neighbors in each layer to minimize the number of message passes. NS-SAGE utilizes this scheme by sampling r neighbors for each node and only collecting the messages from the selected nodes. However, as the number of layers L increases, the complexity of NS-SAGE grows exponentially, making it unsuitable for large graphs with arbitrary numbers of layers. NS-SAGE also requires all neighbors for non-stochastic predictions in the inference phase, resulting in $O(d^L)$ inference time. To address this, VR-GCN [32] employs a variance reduction technique that reduces the size of sampled neighbors. However, VR-GCN adds the memory complexity of $O(Lnf)$ for all nodes' hidden features.

Layer-sampling. Layer-sampling methods perform node sampling in each layer independently, resulting in a constant sample size across all layers and preventing an exponential increase in the neighbor's size. FastGCN [33] uses importance sampling to reduce variance, while Adapt [34] improves FastGCN with an additional sampling network. However, Adapt incurs significant overhead due to the sampling algorithm.

Subgraph-sampling. Certain methods sample a subgraph for each mini-batch and carry out forward and backward passes on the same subgraph across all layers. Cluster-GCN partitions a large graph into multiple densely connected subgraphs and samples a subset of subgraphs (with edges between clusters added back) to train in each mini-batch. Cluster-GCN needs $O(m)$ pre-computation time and $O(bd)$ time to recover the intra-cluster edges when loading each mini-batch. GraphSAINT samples a set of nodes and takes the induced subgraph for mini-batch training. GraphSAINT-RW, the best-performing variant, employs L steps of random walk to induce subgraph from b randomly sampled nodes. Each of the n/b mini-batches covers $O(Lb)$ nodes and edges. Although $O(Ln)$ nodes are sampled with some repetition in an epoch, the number of edges covered is also $O(Ln)$ and is usually much smaller than m. However, like NS-SAGE and some other sampling methods, Cluster-GCN and GraphSAINT-RW cannot make predictions on a randomly sampled subgraph in the inference phase. GraphSAINT-Node, which randomly samples nodes for each mini-batch, does not suffer from this L factor in the complexities, but its performance is inferior to GraphSAINT-RW's.

1.1.3 Large Language Models

Large language models (LLMs) are advanced artificial intelligence systems designed to understand, generate, and reason with human language at large scale. These models, powered by deep learning techniques, are trained on vast datasets of text online, enabling them to grasp and process the knowledge, context, and complexity of natural language. LLMs like OpenAI's GPT series OpenAI [35] have revolutionized fields such as natural language processing, machine translation, and content creation, offering capabilities including answering questions, coding, and generating images and videos. Based on LLMs, a few of effect prompting methodologies are proposed to better exploit the reasoning ability of them. Chain-of-thoughts (CoT) [36] prompting technique has been found useful to enable LLMs towards complex reasoning tasks. By learning to think step by step, LLMs are shown to be capable of tackling more complex and difficult tasks. Building upon CoT, tree-of-thoughts (ToT) [37] enables LLMs to deliberately explore and evaluate reasoning paths for complex problem solving through search over coherent intermediate text "thoughts". Gao et al. [38] proposed program-aided language models (PAL) to let LLMs do reasoning with programs embedded. Zhou et al. [39] proposed Least-to-Most (LTM) prompting to come up with reasoning steps from simple to complex to make reasoning steps more tractable and effective.

Chapter 2: Robust Optimization as Data Augmentation for Large-scale Graphs

2.1 Introduction

Graph Neural Networks (GNNs) have emerged as powerful architectures for learning and analyzing graph representations. The Graph Convolutional Network (GCN) [1] and its variants have been applied to a wide range of tasks, including visual recognition [7], meta-learning [8], social analysis [9, 10], and recommender systems [11]. However, the training of GNNs on large-scale datasets usually suffers from overfitting, and realistic graph datasets often involve a high volume of out-of-distribution test nodes [12], posing significant challenges for prediction problems.

One promising solution to combat overfitting in deep neural networks is data augmentation [13], which is commonplace in computer vision tasks. Data augmentations apply label-preserving transformations to the inputs, such as translations and reflections for images. As a result, data augmentation effectively enlarges the training set while incurring negligible computational overhead. However, it remains an open problem how to effectively generalize the notion of data augmentation to GNNs. Transformations on images rely heavily on image structures [14], and it is challenging to design low-cost transformations that preserve semantic meaning for non-visual tasks like natural language processing [15] and graph learning. Generally speaking, graph data for machine learning comes with graph structure (or edge features) and node features. In the limited

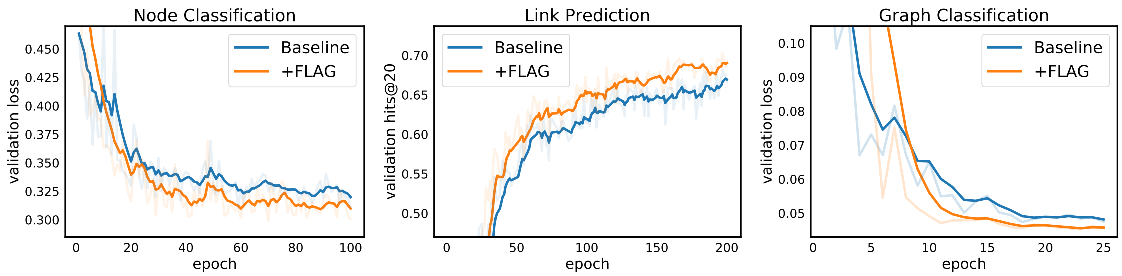


Figure 2.1: Generalization performance of FLAG on all three tasks. Left: node classification with GAT as baseline on `ogbn-products`; Middle: link prediction with hits@20 as metric (the higher the better) and GraphSAGE as baseline on `ogbl-ddi`; Right: graph classification with GIN as baseline on `ogbg-molhiv`. Plotted lines are attained by smoothing the original lines (the shallow ones), where smooth weights are 0.75, 0.75, and 0.5 respectively.

cases where data augmentation can be done on graphs, it generally focuses exclusively on the graph structure by adding/removing edges [16, 17, 18, 19, 20, 21].

In the meantime, adversarial data augmentation, which applies small perturbations in the input feature space to maximially alter model outputs, is known to boost neural network robustness and promote resistance to adversarially chosen inputs [40, 41]. Despite the wide belief that adversarial training harms standard generalization and leads to worse accuracy [42, 43], recently a growing amount of attention has been paid to using adversarial perturbations to augment datasets and ultimately alleviate overfitting. For example, Volpi et al. [44] and Shu et al. [45] showed adversarial data augmentation is a data-dependent regularization that could help generalize to out-of-distribution samples, and its efficacy has been verified in domains including computer vision [46], language understanding [47, 48, 49], and visual question answering [50]. Despite the success of adversarial augmentation in language and vision, it remains unclear how to effectively and efficiently improve GNNs' clean accuracy using adversarial augmentation.

Present work. We propose **FLAG**, **F**ree **L**arge-scale **A**dversarial Augmentation on **G**raphs, to tackle the overfitting problem. While existing literature focuses on modifying graph structures to augment datasets, FLAG works purely in the node feature space by adding adversarial perturbations (generated by gradient-based robust optimization algorithms), to the input node features with graph structures unchanged. FLAG leverages "free" adversarial training methods [51] to conduct efficient adversarial training so that it is highly scalable to large datasets. The method also takes advantage of multi-scale adversarial augmentation to make the model fully generalized in the input feature space. We verify the effectiveness of our method on the *Open Graph Benchmark* (OGB) [12], which is a collection of large-scale, realistic, and diverse graph datasets for node, link, and graph property prediction tasks. We conduct extensive experiments across OGB datasets by applying FLAG to competitive GNN baselines and show that FLAG brings non-trivial improvements in most cases. For example, FLAG lifts the test accuracy of GAT on `ogbn-products` by an absolute value of 2.31%. FLAG is simple (easy to implement with a dozen lines of code in PyTorch), general (model-free and task-free), and efficient (able to bring salient improvement at tractable or even no extra cost). Our main contributions are summarized as follows:

- *Method:* To the best of our knowledge, our work is the *first* general-purpose feature-based data augmentation method on graph data, which is complementary to other regularizers (e.g., dropout) and topological augmentations. The novel method incorporates "free" and multi-scale techniques to craft feature augmentations more effectively.

- *Experiments:* We show the efficacy and scalability of our method through extensive experiments and ablation studies on large-scale datasets across node, link, and graph property prediction tasks. We validate that FLAG is superior to existing adversarial augmentation methods.

- *Analysis:* We provide observations and analysis to support our conjecture that the discrete vs. continuous distribution discrepancy of input features is the key to different effects (beneficial vs. harmful) of adversarial augmentations on model accuracy.

2.2 Preliminaries and Related Work

Graph Neural Networks (GNNs). We denote a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with initial node features x_v for $v \in \mathcal{V}$ and edge features e_{uv} for $(u, v) \in \mathcal{E}$. GNNs are built on graph structures to learn representation vectors h_v for every node $v \in \mathcal{V}$ and a vector $h_{\mathcal{G}}$ for the entire graph $\mathcal{G}$. Following Hu et al. [52], formally the k-th iteration of message passing, or the k-th layer of GNN forward path is defined as:

$$\begin{aligned} msg_v^{(k)} &= \text{AGGREGATE}_\theta^{(k)} \left(\left\{ \left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv} \right), \forall u \in \mathcal{N}(v) \right\} \right) \\ h_v^{(k)} &= \text{COMBINE}_\phi^{(k)} \left(h_v^{(k-1)}, msg_v^{(k)} \right), \end{aligned} \tag{2.1}$$

where $h_v^{(k)}$ is the embedding of node v at the k-th layer, e_{uv} is the feature vector of the edge between node u and v, $\mathcal{N}(v)$ is node v's neighbor set, and $h_v^{(0)} = x_v$. AGGREGATE($\cdot$) and COMBINE($\cdot$) functions are parameterized by neural networks.

To obtain the representation of the entire graph $h_{\mathcal{G}}$, the permutation-invariant READOUT($\cdot$) function pools node features from the final iteration K as:

$$h_{\mathcal{G}} = \text{READOUT} \left(\left\{ h_v^{(K)} \mid v \in \mathcal{V} \right\} \right), \tag{2.2}$$

Existing graph regularizers mainly focus on augmenting graph structures by modifying edges [16, 17, 33]. GraphAT [53], BVAT [54], and LAT [55] are three *semi-supervised* methods on the node classification task. GraphAT promotes local smoothness by reinforcing the similarity between the predictions of perturbed nodes and their neighbors. BVAT proposed two graph VAT schemes to enhance the output smoothness of GCN; LAT virtually perturbed the first-layer embedding of a GCN classifier. The usage scenario of these methods is limited to node classification, while data augmentation should function regardless of tasks. Besides, the formulation of VAT [56] utilized by these works involves both supervised clean and adversarial robust losses simultaneously. Practically this will consume at least twice the GPU memory as the baseline, making them not scalable to large-scale datasets. Overall, no work so far has considered general-purpose feature-based data augmentations for large-scale graphs.

2.3 Proposed Method

In this work, we investigate how to effectively improve the generalization of GNNs through a feature-based augmentation. Graph node features are usually constructed as discrete embeddings, such as binary bag-of-words vectors or categorical variables. As a result, standard hand-crafted augmentations, like flipping and cropping transforms used in computer vision, are not applicable to graphs node features.

By hunting for and stamping out small perturbations that cause the classifier to fail, one may hope that adversarial training could benefit standard accuracy [40, 42, 56]. It is widely observed that when the data distribution is sparse and discrete, the beneficial effect of adversarial perturbations on generalization takes over [42, 50]. Volpi et al. [44] viewed adversarial perturbation as a data-dependent regularization, which could intuitively generalize to out-of-distribution samples. Highlighted by Hu et al. [12], the out-of-distribution phenomenon of data is salient in the graph domain, and also considering the sparsity of labeled node samples in the semi-supervised node classification task, we view adversarial perturbation as a strong candidate method for input feature augmentation.

Min-Max Optimization. Adversarial training is the process of crafting adversarial data points, and then injecting them intro training data. This process is often formulated as the following min-max problem:

$$\min_{\boldsymbol{\theta}} \quad E_{(x,y)\sim\mathcal{D}} \left[\max_{\|\boldsymbol{\delta}\|_p \leq \epsilon} L\left(f_{\boldsymbol{\theta}}(x+\boldsymbol{\delta}), y\right) \right], \tag{2.3}$$

where $\mathcal{D}$ is the data distribution, y is the label, $\|\cdot\|_p$ is some ℓ_p-norm distance metric, ϵ is the perturbation budget, and L is the objective function. Madry et al. [41] showed that this saddle-point optimization problem could be reliably tackled by Stochastic Gradient Descent (SGD) for the outer minimization and Projected Gradient Descent (PGD) for the inner maximization. In practice, the typical approximation of the inner maximization under an l_∞-norm constraint is as follows,

$$\boldsymbol{\delta}_{t+1} = \pi_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \left(\boldsymbol{\delta}_t + \alpha \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{\delta}} L\left(f_{\boldsymbol{\theta}}(x+\boldsymbol{\delta}_t), y\right)\right)\right), \tag{2.4}$$

where the perturbation $\boldsymbol{\delta}$ is updated iteratively, and $\pi_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon}$ performs projection onto the ϵ-ball in the l_∞-norm. For maximum robustness, this iterative updating procedure usually loops M times to craft the worst-case noise, which requires M forward and backward passes end-to-end. Afterwards the most vicious noise $\boldsymbol{\delta}_M$ is applied to the input feature, on which the model weight is optimized. The algorithm above is called PGD.

Multi-scale Augmentation. On visual tasks, Chen et al. [14] highlighted the importance of using *diverse* types of data augmentations such as *random cropping*, *color distortion*, and *Gaussian blur*. The authors showed that a single transformation is not sufficient to learn good

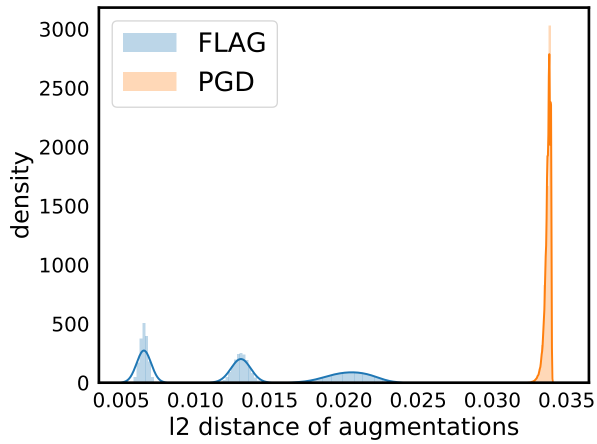


Figure 2.2: Augmentation distance distributions of FLAG and PGD. We run the test on `ogbn-arxiv` with GCN as backbone. Ascent steps are both set to 3.

representations. To fully exploit the generalizing ability and enhance the diversity and quality of adversarial perturbations, we propose to craft multi-scale augmentations. To realize this goal, we leverage the techniques below.

"Free" training. We leverage "free" adversarial training [51] to craft adversarial data augmentations. PGD is a powerful yet inefficient way of solving the min-max optimization. It runs M full forward and backward passes to craft a refined perturbation $\boldsymbol{\delta}_{1:M}$, but the model weights $\boldsymbol{\theta}$ only get updated once using the final $\boldsymbol{\delta}_M$. This process makes model training M times slower. In contrast, while computing the gradient for the perturbation $\boldsymbol{\delta}$, "free" training simultaneously produces the model parameter $\boldsymbol{\theta}$ on the same backward pass. This enables a parameter update to be computed in parallel with a perturbation update at virtually no additional cost. The authors proposed to train on the same minibatch M times in a row to simulate the inner maximization in Eq. (2.3), while compensating by performing M times fewer epochs of training. The resulting algorithm yields accuracy and robustness competitive with standard adversarial training, but with the same runtime as clean training.

Besides the efficiency, the "free" method achieves our idea of optimizing $\boldsymbol{\theta}$ with multi-scale augmentations. Note that X is augmented with additive perturbations $\boldsymbol{\delta}_{1:M}$, of which each can have a maximum scale of $m\alpha, m \in \{1, \cdots, M\}$, in contrast to PGD whose perturbation is a single $\boldsymbol{\delta}_M$ with an $M\alpha$ scaling. This greatly adds to the diversity of our augmentations. However, the "free" algorithm is suboptimal in terms of min-max optimization in that during the batch-replay process, the approximated perturbation computed to maximize the objective on $\boldsymbol{\theta_t}$ is used to robustly optimize $\boldsymbol{\theta_{t+1}}$ rather than $\boldsymbol{\theta_t}$. To tackle this problem, instead of directly updating $\boldsymbol{\theta}$ using the "by-product" gradient attained from the gradient ascent step on $\boldsymbol{\delta}$, we accumulate the gradients $\nabla_{\boldsymbol{\theta}} L$, and apply them to the model parameters all at once later. Formally, the optimization step is

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \frac{\tau}{M} \sum_{t=1}^{M} \nabla_{\boldsymbol{\theta}} L \left(f_{\boldsymbol{\theta}}(x + \boldsymbol{\delta_t}), y \right), \tag{2.5}$$

where τ is learning rate and $\boldsymbol{\delta}_1$ is uniform noise. Note that the gradients in Eq.(2.5) are restored when crafting perturbation in Eq.(2.4). We save one backward pass and M times extra GPU memory through accumulating gradients (which is fully supported by PyTorch) during the batch replay process. Figure 2.2 depicts the effects of our design. We can see that PGD inevitably produces concentrated augmentations in terms of the magnitude, whereas our method produces perturbations with a broader range of sizes, which adds to the diversity and quality of the augmentations.

Moreover on the node classification task, we propose to augment labeled vs. unlabeled nodes with diverse magnitudes of perturbations during training time to further diversify the augmentations. We call it *Weighted perturbation*. When classifying one target node, messages from the whole k-hop neighborhood are aggregated and combined into its embedding. It is natural to believe that a further neighbor should have lower impact, i.e. higher smoothness, on the final decision of the target node, which can also be intuitively reflected by the recursive message passing

Algorithm 1: **FLAG**: **F**ree **L**arge-scale **A**dversarial Augmentation on **G**raphs (Node Classification Task)

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{V}_l$ is the labeled node set; learning rate τ; ascent steps M; ascent step size α_v for labeled node, α_u for unlabeled, we assume the neighbors of labeled nodes are all unlabeled ones; $L(\cdot)$ as objective function; A$(\cdot)$ and C$(\cdot)$ denote the AGGREGATE and COMBINE functions in Eq.(4.1). The backward function at line 12 refers to back-propagation gradient computation for both model weights and noises.

1 Initialize (θ, ϕ)
2 **for** $v \in \mathcal{V}_l$ **do**
3 $\quad \delta_v^{(0)} \leftarrow U(-\alpha_v, \alpha_v)$
4 $\quad \delta_u^{(0)} \leftarrow U(-\alpha_u, \alpha_u)$
5 $\quad$ **for** t $= 1 \ldots M$ **do**
6 $\quad\quad h_v^{(0)} \leftarrow x_v + \delta_v^{(t-1)}$
7 $\quad\quad h_u^{(0)} \leftarrow x_u + \delta_u^{(t-1)}$
8 $\quad\quad$ **for** $k = 1 \ldots K$ **do**
9 $\quad\quad\quad msg_v^{(k)} \leftarrow \mathrm{A}_\theta^{(k)}\left(\left\{\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}\right), \forall u \in \mathcal{N}(v)\right\}\right)$
10 $\quad\quad\quad h_v^{(k)} \leftarrow \mathrm{C}_\phi^{(k)}\left(h_v^{(k-1)}, msg_v^{(k)}\right)$
11 $\quad\quad$ **end for**
12 $\quad\quad L\left(h_v^{(K)}, y\right)$.backward()
13 $\quad\quad g_{\theta,\phi}^{(t)} \leftarrow g_{\theta,\phi}^{(t-1)} + \frac{1}{M} \cdot \mathrm{grad}(\theta, \phi)$
14 $\quad\quad \delta_v^{(t)} \leftarrow \delta_v^{(t-1)} + \alpha_v \cdot \mathrm{sign}\left(\mathrm{grad}\left(\delta_v\right)\right)$
15 $\quad\quad \delta_u^{(t)} \leftarrow \delta_u^{(t-1)} + \alpha_u \cdot \mathrm{sign}\left(\mathrm{grad}\left(\delta_u\right)\right)$
16 $\quad$ **end for**
17 $\quad (\theta, \phi) \leftarrow (\theta, \phi) - \tau \cdot g_{\theta,\phi}^{(M)}$
18 **end for**

procedure of GNNs in Eq.(4.1). In practice we find that a larger perturbation for unlabeled nodes can be beneficial to the performance. Algorithm 1 summarizes the pseudo code of our method on node classification task. Figure 2.1 illustrates the generalization ability of our proposed method.

2.4 Experiments

In this section, we conduct extensive experiments to fully reveal the efficacy of our method.

Datasets. We demonstrate FLAG's effectiveness through extensive experiments on the *Open Graph Benchmark* (OGB), which consists of a wide range of challenging large-scale datasets. Shchur et al. [57], Errica et al. [58], and Dwivedi et al. [59] showed that traditional graph datasets suffered from problems such as unrealistic and arbitrary data splits, highly limited data sizes, non-rigorous evaluation metrics, and common neglect of validation set, etc. In order to empirically study FLAG's effects in a fair and reliable manner, we conduct experiments on the OGB [12] datasets, which have tackled those major issues and brought more realistic challenges to the graph research community.

Setup. FLAG drops the projection step when performing the inner maximization, in light of the positive effect of large perturbations on generalization [44], and also to simplify hyperparameter search. Usually on images, the inner maximization has a norm constraint on the perturbation; the largest perturbation one can add is bounded by the hyperparameter ϵ, typically 8/255 under the l_∞-norm. This ϵ encourages the visual imperceptibility of the perturbations, thus making defenses realistic and practical. However, graph node features or language word embeddings do not have an established distance threshold for imperceptibility, which makes the selection of ϵ highly heuristic. Note that, although the perturbation is no longer bounded by an explicit ϵ in FLAG, it is still implicitly bounded in the furthest distance that $\boldsymbol{\delta}$ can reach, i.e. the step size α times the number of ascending steps M.

Also unless otherwise stated, all of the baseline test statistics come from the official OGB leaderboard website, and we conduct all of our experiments using publicly released implementations without touching the original model architecture or training setup for fair comparisons. We report mean and standard deviations from 10 runs with different random seeds. Following common practice on this benchmark, we report the test performance associated with the best validation

Table 2.1: Node property prediction test performance on `ogbn-products`, `ogbn-proteins`, and `ogbn-arxiv` datasets. Blank denotes no statistics on the leaderboard.

Backbone	`ogbn-products` Test Acc	`ogbn-proteins` Test ROC-AUC	`ogbn-arxiv` Test Acc
GCN	-	**72.51**±**0.35**	71.74±0.29
+FLAG	-	71.71±0.50	**72.04**±**0.20**
GraphSAGE	78.70±0.36	**77.68** ±**0.20**	71.49±0.27
+FLAG	**79.36**±**0.57**	76.57±0.75	**72.19**±**0.21**
GAT	79.45±0.59	-	73.65±0.11
+FLAG	**81.76**±**0.45**	-	**73.71**±**0.13**
DeeperGCN	80.98±0.20	85.80±0.17	71.92±0.16
+FLAG	**81.93**±**0.31**	**85.96**±**0.27**	**72.14**±**0.19**

result. We choose GCN, GraphSAGE, GAT, and GIN as our baseline models. In addition, we apply FLAG to the DeeperGCN model to demonstrate its effectiveness on the GNNs with significantly deeper depth. Our implementation always uses $M = 3$ ascent steps for simplicity. Following Goodfellow et al. [40], Madry et al. [41], we use $\text{sign}(\cdot)$ for gradient normalization.

Large-scale Node Property Prediction. We summarize the results of node classification in Table 2.1. Notably, FLAG yields a 2.31% test accuracy lift for GAT, making GAT competitive on the `ogbn-products` dataset. Considering the specialty of not having input node features in `ogbn-proteins`, we provide detailed discussions on the effect of different node feature constructions in Section 4.5. `ogbn-mag` is a heterogeneous network where only "paper" nodes come with node features. We use the neighbor sampling mini-batch algorithm to train R-GCN and report its results in the Table 2.2. Surprisingly, FLAG can also directly bring nontrivial accuracy improvement without special designs for heterogeneous graphs, which demonstrates its versatility.

Table 2.2: Test performance on the heterogeneous OGB node property prediction dataset `ogbn-mag`.

Backbone	`ogbn-mag` Test Acc
R-GCN	46.78±0.67
+FLAG	**47.37**±**0.48**

Table 2.3: Link property prediction test performance on `ogbl-ddi` and `ogbl-collab` datasets.

Backbone	`ogbl-ddi` Hits@20	`ogbl-collab` Hits@50
GCN	37.07 ±5.07	44.75±1.07
+FLAG	**51.41±3.76**	**46.22±0.81**
GraphSAGE	53.90 ±4.74	48.10 ±0.81
+FLAG	**63.31±6.06**	**48.44±0.40**

Table 2.4: Graph property test performance on `ogbg-molhiv`, `ogbg-molpcba`, `ogbg-ppa`, and `ogbg-code` datasets. ♮ denotes the existence of virtual nodes; blank denotes no statistics on the leaderboard.

Backbone	`ogbg-molhiv` Test ROC-AUC	`ogbg-molpcba` Test AP	`ogbg-ppa` Test Acc	`ogbg-code` Test F1
GCN	76.06±0.97	20.20±0.24	68.39±0.34	31.63±0.18
+FLAG	**76.83±1.02**	**21.16±0.17**	68.38±0.47	**32.09±0.19**
GCN-Virtual	**75.99±1.19**	24.24±0.34	68.57±0.61	32.63±0.13
+FLAG	75.45±1.58	**24.83±0.37**	**69.44±0.52**	**33.16±0.25**
GIN	75.58±1.40	22.66±0.28	68.92±1.00	31.63±0.20
+FLAG	**76.54±1.14**	**23.95±0.40**	**69.05±0.92**	**32.41±0.40**
GIN-Virtual	77.07±1.49	27.03±0.23	70.37±1.07	32.04±0.18
+FLAG	**77.48±0.96**	**28.34±0.38**	**72.45±1.14**	**32.96±0.36**
DeeperGCN	78.58±1.17	27.81♮±0.38	77.12±0.71	-
+FLAG	**79.42±1.20**	**28.42♮±0.43**	**77.52±0.69**	-

Large-scale Link Property Prediction. We evaluate our method on two OGB link prediction datasets, which are `ogbl-ddi` and `ogbl-collab`. The authors of OGB selected Hits@K as the official evaluation metric. We study the performance of FLAG with GCN and GraphSAGE as backbone on this task. We follow the practice of the baselines to train the models in the full-batch manner. Results are reported in Table 2.3. We highlight that FLAG brings a salient boost to both GCN and GraphSAGE on the `ogbl-ddi` dataset.

Large-scale Graph Property Prediction. Table 2.4 summarizes the test scores of GCN, GIN, and DeeperGCN on all four OGB graph property prediction datasets. "Virtual" means the model is augmented with virtual nodes [12, 18, 60]. As adversarial perturbations are crafted by gradient ascent, it would be unnatural and suboptimal to add noises to discrete input node features [61]. We firstly project discrete node features into the continuous space and then adversarially augment the hidden embeddings. On `ogbg-molhiv`, FLAG yields notable improvements, but when GCN has already been hurt by virtual nodes, FLAG appears to exaggerate the harm. On `ogbg-molpcba`, GIN-Virtual with FLAG receives an absolute value 1.31% test AP value increase. Besides node classification and link prediction, FLAG's strong effects on graph classification prove its high versatility.

2.5 Ablation Studies and Discussions

Compatibility with graph structure regularizers. As our augmentation manipulates the input features, it is highly complementary to structure-based regularizers. We validate this point through the experiments below. We mainly focus on two widely-used topological augmentation methods to illustrate [1]: (i) Neighbor sampling [17] randomly samples neighbors for information aggregation. It not only contributes to GNN scalability but also acts as a structure regularizer. A full-batch GraphSAGE reaches $78.50 \pm 0.14\%$ test accuracy on `ogbn-products`, and neighbor sampling alone generalizes the model to $78.70 \pm 0.36\%$. When FLAG is also used, the test accuracy is increased to $79.36 \pm 0.57\%$. (ii) Virtual node [18] adds one synthetic node that connects to all existing nodes. Nearly all the numbers from Table 2.4 supports that our method works well with virtual node to generalize GNNs further. Here We highlight one representative group of experiments on `ogbg-ppa` with GIN as baseline. Vanilla GIN gets $68.92 \pm 1.00\%$ test accuracy. By adding virtual node alone, it goes to $70.37 \pm 1.07\%$. When FLAG is further deployed, test

[1] We also tried DropEdge [16] but it failed to yield performance gain in the first place.

Table 2.5: Test Accuracy on the `ogbn-arxiv` dataset with different BN methods.

Method	GCN	GraphSAGE
w/o BN	71.09±0.22	69.58±0.76
w/ BN	71.74±0.29	71.49±0.27
w/ BN +FLAG	72.04±0.20	72.19±0.21
w/ Dual BN +FLAG	**72.11**±**0.23**	**72.21**±**0.20**

Table 2.6: Test performances on different datasets trained with different adversarial augmentations. Baselines are GAT, GraphSAGE, and GIN respectively. FLAG (fast) means the training epoch number is decreased to make our method trained as fast as the baseline.

	`ogbn-products` Test Acc	`ogbl-ddi` Hits@20	`ogbg-molhiv` Test ROC-AUC
Baseline	79.45±0.59	53.90±4.74	75.58±1.40
+PGD	80.96±0.41	62.02±6.56	76.14±1.62
+“Free”	79.42±0.84	58.61±6.0	74.93±1.29
+FLAG	**81.76**±**0.45**	**63.31**±**6.06**	**76.54**±**1.14**
+FLAG (fast)	80.64±0.74	-	-

accuracy reaches $72.45 \pm 1.14\%$.

Compatibility with batch norm. Batch norm is appearing more and more frequently in top-performing GNNs. Xie et al. [46] argued that there was a potential risk, that adversarial examples could distort BN parameters away from natural distribution so to cause the adversarially trained model to fail on clean samples. The authors proposed to use dual batch norms (one for adversaries and the other for clean ones) at training time to better exploit the generalization ability of adversarial augmentations. To test the dual batch norm method on graph data, we run experiments as summarized in Table 2.5. We find that the utilization of dual BN can produce a slight performance gain. As there is growing attention on using batch norms on GNNs, it will be interesting to see how to better synergize adversarial augmentation with batch norms in future research.

Comparison with other robust optimization methods. Table 2.6 shows performances with different adversarial augmentations. For PGD and “free”, we compute 8 ascent steps for

Table 2.7: Test Accuracy on the `ogbn-products` dataset.

Backbone	Test Acc
GAT w/o dropout	75.67±0.27
GAT w/ dropout	79.45±0.59
GAT w/ dropout +FLAG	**81.76±0.45**

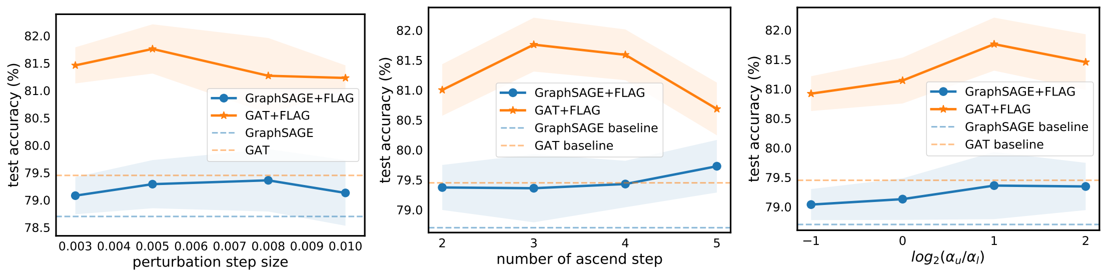
Table 2.8: Test accuracy on `ogbn-products` with GraphSAGE trained with diverse mini-batch algorithms.

Backbone	`ogbn-products` Test Acc
GraphSAGE w/ NS	78.70±0.36
+FLAG	**79.36±0.57**
GraphSAGE w/ Cluster	**78.97±0.33**
+FLAG	78.60±0.27
GraphSAGE w/ SAINT	79.08±0.24
+FLAG	**79.60±0.19**

the inner-maximization to make the attack strong enough, while for FLAG we only compute 3 steps. We can see that FLAG outperforms all other methods. We attribute that to the practice of our multi-scale augmentation, which diversifies the scale range of feature perturbations, and helps the model see diverse input features to generalize better, especially on out-distribution samples. Although "free" method incorporates diversifying augmentations, but here the benefits are overwhelmed by the suboptimal problem.

Effects of weighted perturbation. The effects of biased perturbation are reported in Figure 2.3(c). Generally speaking, when $\log_2(\alpha_u/\alpha_l) > 0$, which means that unlabeled nodes receive larger augmentations, the performance gains are more salient. The phenomenon supports our practice of using weighted perturbation to promote multi-scale augmentations. Empirically we find that the benefit of weighted perturbation is more evident on `ogbn-products` than on `ogbn-arxiv`. Our understanding is that, `ogbn-products` is better suited with our practice of labeled vs. unlabeled split because of its high label sparsity compared with `ogbn-arxiv` (label rate 8% vs. 54%). When labeled nodes are more sparse, the neighborhood of labeled nodes will be more overwhelmed by unlabeled ones, where our approximation is more accurate.

Hyperparameter sensitivity. Figure 2.3(a) and Figure 2.3(b) show the hyperparameter sensitivity of our method. Overall, our method is stable to yield consistent accuracy boost compared with baseline.



((a)) step size ((b)) ascent steps ((c)) weighted perturbation

Figure 2.3: Results of GraphSAGE and GAT on the `ogbn-products` dataset.

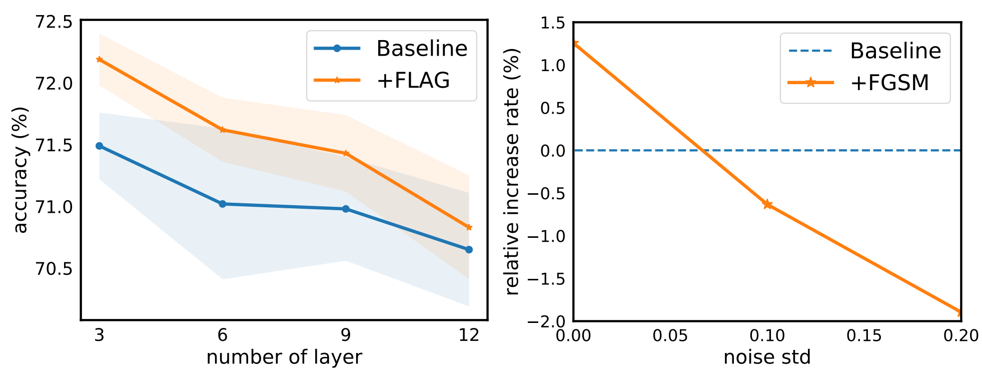
Compatibility with mini-batch methods. Graph mini-batch algorithms are critical to training GNNs on large-scale datasets. We test how different algorithms will work with adversarial data augmentation with GraphSAGE as the backbone. From Table 2.8, we see that neighbor sampling [17] and GraphSAINT [31] can all work with FLAG to further boost performance, while Cluster [30] suffers an accuracy drop.

Compatibility with dropout. Dropout is widely used in GNNs. Table 2.7 shows that, when trained without dropout, GAT accuracy drops steeply by a large margin. What is more, FLAG can further generalize GNN models together with dropout, similar to the phenomenon of image augmentations. It demonstrates that our method is fully compatible with this domain/model-agnostic regularizer.

Towards going "free". FLAG introduces tractable extra training overhead. We empirically show that, when we decrease the total number of training epochs to make it as fast as the standard

GNN training pipeline, FLAG still brings significant performance gains. Table 2.6 shows that FLAG with fewer epochs still generalizes the baseline. Empirically, on a single Nvidia RTX 2080Ti, 100-epoch vanilla GAT takes 88 mins, while FLAG (fast) in Table 2.6 takes 91 mins. We note that heuristics like early stopping and cyclic learning rates can further accelerate the adversarial training process [62], so there are abundant opportunities for further research on adversarial augmentation at lower or even no cost.

Towards going deep. Over-smoothing stops GNNs from going deep. FLAG shows its ability to boost both shallow and deep baselines, e.g., GCN and DeeperGCN. We carefully examine FLAG's effects on generalization when a GNN goes progressively deeper in Figure 2.4(a). The experiments are conducted on `ogbn-arxiv` with GraphSAGE as the backbone, where a consistent improvement is evident.



((a)) ((b))

Figure 2.4: (a) Test accuracy on `ogbn-arxiv`; (b) Performance gap on `Cora`

What if there's no node feature? One natural question can be raised: what if no input node features are provided? `ogbn-proteins` is a dataset without input node features. Hu et al. [12] proposed to average incoming edge features to obtain initial node features, while [63] used summation and achieved competitive results. Note that the GCN and GraphSAGE baselines in Table 2.1 use the "mean" node features as input and suffer an accuracy drop with FLAG;

DeeperGCN leverages the "sum" and gets further improved. Interestingly, when DeeperGCN is trained with "mean" node features, it receives high invariance, so that even large magnitude perturbations will not change its result. The diverse behavior of adversarial augmentation implies the importance of node feature construction method selection.

2.6 Conclusion

We propose FLAG, a simple, scalable, and general data augmentation method for better GNN generalization. Like widely-used image augmentations, FLAG can be easily incorporated into any GNN training pipeline. FLAG yields improvements over a range of GNN baselines. Besides extensive experiments, we also provide conceptual analysis to validate adversarial augmentation's different behavior on varied data types. The effects of adversarial augmentation on generalization are still not entirely understood, and we think this is a fertile space for future exploration.

2.7 Appendix

We summarize implementation details and selected hyperparameters in this section. Note that for ALL of our method, we fix the ascent step number M to 3 for simplicity. We leave more thorough step number search for future research. Experiments are done on hardware with Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, and 128GB of RAM. If without mentioning, we use a single GeForce RTX 2080 Ti (11GB GPU memory). To highlight, for fair comparisons, we do not modify model architectures nor optimizing algorithms.

```python
#M as ascent steps, alpha as ascent step size
#X denotes input node features, y denotes labels
def flag(gnn, X, y, optimizer, criterion, M, alpha) :
    gnn.train()
    optimizer.zero_grad()

    pert = torch.FloatTensor(*X.shape).uniform_(-alpha, alpha)
    pert.requires_grad_()
    out = gnn(X+pert)
    loss = criterion(out, y)/M

    for _ in range(M-1):
        loss.backward()
        pert_data = pert.detach() + alpha*torch.sign(pert.grad.detach())
        pert.data = pert_data.data
        pert.grad[:] = 0
        out = gnn(X+pert)
        loss = criterion(out, y)/M

    loss.backward()
    optimizer.step()
```

Figure 2.5: An abstract PyTorch Implementation of our method.

2.7.1 Node Classification

2.7.1.1 ogbn-products

MLP: perturbation step size α=2e-02, only labeled nodes are used in the training phase.

GraphSAGE: labeled perturbation step size α_l=8e-03, α_u/α_l=2, and neighbor sampling is used for scalable training.

GAT: labeled perturbation step size α_l=5e-03, α_u/α_l=2, and neighbor sampling is used for scalable training.

DeeperGCN: labeled perturbation step size α_l=5e-03, α_u/α_l=2, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.1.2 ogbn-proteins

GCN: labeled perturbation step size α_l=1e-03, α_u/α_l=1, and the model is trained in the full-batch manner.

GraphSAGE: labeled perturbation step size α_l=1e-03, α_u/α_l=1, and the model is trained in the full-batch manner.

DeeperGCN: labeled perturbation step size α_l=8e-03, α_u/α_l=1, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.1.3 ogbn-arxiv

MLP: perturbation step size α=2e-03, only labeled nodes are used in the training phase.

GCN: labeled perturbation step size α_l=1e-03, α_u/α_l=1, and the model is trained in the full-batch manner.

GraphSAGE: labeled perturbation step size α_l=1e-03, α_u/α_l=1, and the model is trained in the full-batch manner.

GAT: labeled perturbation step size α_l=1e-03, α_u/α_l=2, and the model is trained in the full-batch manner.

DeeperGCN: labeled perturbation step size α_l=8e-03, α_u/α_l=1, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.1.4 ogbn-mag

R-GCN: labeled perturbation step size α_l=1e-04, α_u/α_l=1, and the model is trained with neighbor sampling for scalability.

2.7.2 Link Prediction

2.7.2.1 ogbl-ddi

GCN perturbation step size α=3e-03, and **GraphSAGE** perturbation step size α_l=3e-03. Models are both trained in the full-batch manner. During each gradient ascent loop negative edges are resampled for computing negative losses.

2.7.2.2 ogbl-collab

GCN perturbation step size α=3e-03, and **GraphSAGE** perturbation step size α_l=3e-03. Models are both trained in the full-batch manner. During each gradient ascent loop negative edges are resampled for computing negative losses.

2.7.3 Graph Classification

2.7.3.1 ogbg-molhiv

GCN: perturbation step size α=1e-02, when virtual node is added we use a smaller α=1e-03.

GIN: perturbation step size α=5e-03, when virtual node is added we use a smaller α=1e-03.

DeeperGCN: perturbation step size α=1e-02, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.3.2 ogbg-molpcba

GCN: perturbation step size α=8e-03 for both the vanilla model and the one augmented by virtual node.

GIN: perturbation step size α=8e-03 for both the vanilla model and the one augmented by virtual node.

DeeperGCN: perturbation step size α=8e-03 with virtual node added, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.3.3 ogbg-ppa

GCN: perturbation step size α=2e-03, when virtual node is added we use a larger α=5e-03.

GIN: perturbation step size α=8e-03, when virtual node is added we use a smaller α=5e-03.

DeeperGCN: perturbation step size α=8e-03, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.3.4 ogbg-code

GCN: perturbation step size α=8e-03 for both the vanilla model and the one augmented by virtual node.

GIN: perturbation step size α=8e-03 for both the vanilla model and the one augmented by virtual node.

DeeperGCN: perturbation step size α=8e-03 with virtual node added, and the model is trained on NVIDIA Tesla V100 (32GB GPU memory).

2.7.4 Dataset Details

Table 2.9, Table 2.10, and Table 2.11 summarize the dataset statistics for node classification, link prediction, and graph classification respectively. Datasets can be downloaded at `https:`

Name	#Nodes	#Edges	#Tasks	Train/Val/Test	Task Type	Metric
ogbn-products	2,449,029	61,859,140	1	8/2/90	Multi-class classification	Accuracy
ogbn-proteins	132,534	39,561,252	112	65/16/19	Binary classification	ROC-AUC
ogbn-arxiv	169,343	1,166,243	1	54/18/28	Multi-class classification	Accuracy
ogbn-mag	1,939,743	21,111,007	1	85/9/6	Multi-class classification	Accuracy
Cora	2485	5069	1	no official split$*$	Multi-class classification	Accuracy

Table 2.9: Node classification datasets statistics. $*$ denotes we follow the split of Kipf and Welling [1].

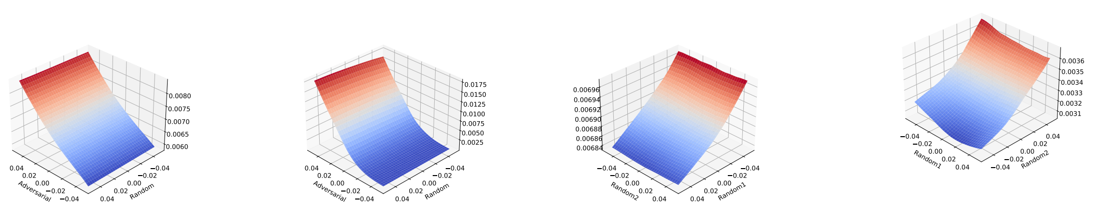
Name	#Nodes	#Edges	Train/Val/Test	Task Type	Metric
ogbl-ddi	4,267	1,334,889	80/10/10	Link prediction	Hits@20
ogbl-collab	235,868	1,285,465	92/4/4	Link prediction	Hits@50

Table 2.10: Link prediction datasets statistics.

//ogb.stanford.edu/.

2.7.5 Loss Landscape Visualization

Figure 2.6 shows the loss landscape of GIN model. We can see that our method further regularizes the loss landscape.



((a)) FLAG adv-random ((b)) Vanilla adv-random ((c)) FLAG random-random ((d)) Vanilla random-random

Figure 2.6: Loss landscape visualization. The test is conducted on one random validation graph from `ogbg-molhiv`. Two models are GIN trained with FLAG and a vanilla GIN. (a) and (b) projects loss onto a random direction and the other adversarial direction, while (c) and (d) use two random directions.

Name	#Graphs	Avg #Nodes	Avg #Edges	#Tasks	Train/Val/Test	Task Type	Metric
ogbg-molhiv	41,127	25.5	27.5	1	80/10/10	Binary classification	ROC-AUC
ogbg-molpcba	437,929	26.0	28.1	128	80/10/10	Binary classification	AP
ogbg-ppa	158,100	243.4	2,266.1	1	49/29/22	Multi-class classification	Accuracy
ogbg-code	452,741	125.2	124.2	1	90/5/5	Sub-token prediction	F1 score

Table 2.11: Graph classification datasets statistics.

Chapter 3: VQ-GNN: A Universal Framework to Scale-up Graph Neural Networks using Vector Quantization

3.1 Introduction

The rise of Graph Neural Networks (GNNs) has brought the modeling of complex graph data into a new era. Using message-passing, GNNs iteratively share information between neighbors in a graph to make predictions of node labels, edge labels, or graph-level properties. A number of powerful GNN architectures [1, 17, 28, 29] have been widely applied to solve down-stream tasks such as recommendation, social analysis, visual recognition, etc.

With the soaring size of realistic graph datasets and the industrial need to model them efficiently, GNNs are hindered by a scalability problem. An L-layer GNN aggregates information from all L-hop neighbors, and standard training routines require these neighbors to all lie on the GPU at once. This prohibits full-batch training when facing a graph with millions of nodes [12].

A number of sampling-based methods have been proposed to accommodate large graphs with limited GPU resources. These techniques can be broadly classified into three categories: (1) Neighbor-sampling methods [17, 32] sample a fixed-number of neighbors for each node; (2) Layer-sampling methods [33, 64] sample nodes in each layer independently with a constant sample size; (3) Subgraph-sampling methods [30, 31] sample a subgraph for each mini-batch and

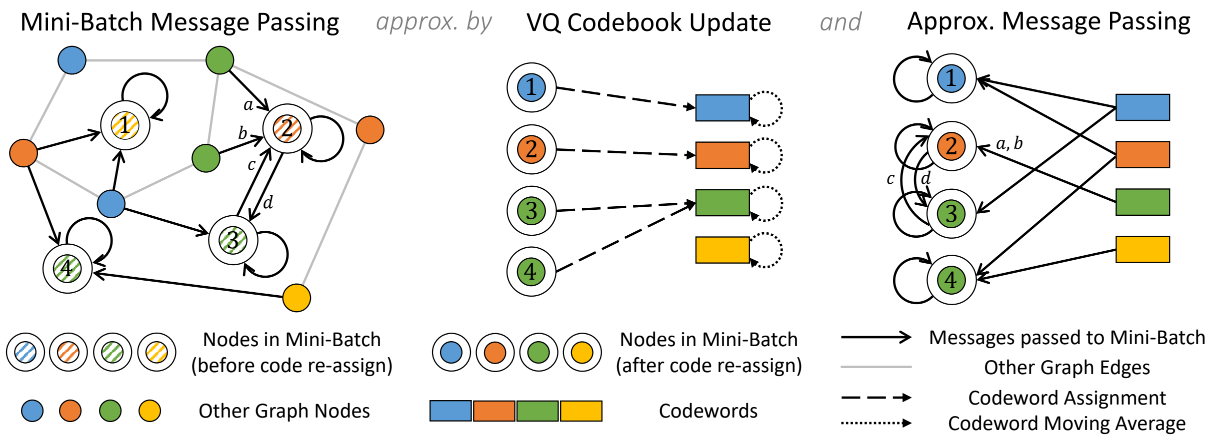


Figure 3.1: In our framework, VQ-GNN, each *mini-batch message passing* (left) is approximated by a *VQ codebook update* (middle) and an *approximated message passing* (right). All the messages passed to the nodes in the current mini-batch are effectively preserved. Circles are nodes, and rectangles are VQ codewords. A double circle indicates nodes in the current mini-batch. Color represents codeword assignment. During *VQ codebook update*, codeword assignment of nodes in the mini-batch is refreshed (node 1), and codewords are updated using the assigned nodes. During *approximated message passing*, messages from out-of-mini-batch nodes are approximated by messages from the corresponding codewords, messages from nodes assigned to the same codeword are merged (a and b), and intra-mini-batch messages are not changed (c and d).

perform forward and back-propagation on the same subgraph across all layers. Although these sampling-based methods may significantly speed up the training time of GNNs, they suffer from the following three major drawbacks: (1) At inference phase, sampling methods require all the neighbors to draw non-stochastic predictions, resulting in expensive predictions if the full graph cannot be fit on the inference device; (2) As reported in [12] and in Section 3.6, state-of-the-art sampling-baselines fail to achieve satisfactory results consistently across various tasks and datasets; (3) Sampling-based methods cannot be universally applied to GNNs that utilize many-hop or global context in each layer, which hinders the application of more powerful GNNs to large graphs.

This paper presents VQ-GNN, a GNN framework using vector quantization to scale most state-of-the-art GNNs to large graphs through a principled and fundamentally different approach compared with the sampling-based methods. We explore the idea of using vector quantization (VQ) as a means of dimensionality reduction to learn and update a small number of quantized reference vectors (codewords) of global node representations. In VQ-GNN, mini-batch message passing in each GNN layer is approximated by a VQ codebook update and an approximated form of message passing between the mini-batch of nodes and codewords; see Fig. 3.1. Our approach avoids the "neighbor explosion" problem and enables mini-batch training and inference of GNNs. In contrast to sampling-based techniques, VQ-GNN can effectively preserve all the messages passed to a mini-batch of nodes. We theoretically and experimentally show that our approach is efficient in terms of memory usage, training/inference time, and convergence speed. Experiments on various GNN backbones demonstrate the competitive performance of our framework compared with the full-graph training baseline and sampling-based scalable algorithms.

Paper organization. The remainder of this paper is organized as follows. Section 3.2 summarizes GNNs that can be re-formulated into a common framework of graph convolution. Section 3.3 defines the scalability challenge of GNNs and shows that dimensionality reduction is a potential solution. In Section 4.3, we describe our approach, VQ-GNN, from theoretical framework to algorithm design and explain why it solves the scalability issue of most GNNs. Section 3.5 compares our approach to the sampling-based methods. Section 3.6 presents a series of experiments that validate the efficiency, robustness, and universality of VQ-GNN. Finally, Section 3.7 concludes this paper with a summary of limitations and broader impacts.

3.2 Preliminaries: GNNs defined as Graph Convolution

Notations. Consider a graph with n nodes and m edges (average degree $d = m/n$). Connectivity is given by the adjacency matrix $A \in \{0, 1\}^{n\times n}$ and features are defined on nodes by $X \in \mathbb{R}^{n\times f_0}$ with f_0 the length of feature vectors. Given a matrix C, let $C_{i,j}$, $C_{i,:}$, and $C_{:,j}$ denote its (i, j)-th

Table 3.1: Summary of GNNs re-formulated as generalized graph convolution.

Model Name	Design Idea	Conv. Matrix Type	# of Conv.	Convolution Matrix
GCN[1] [1]	Spatial Conv.	Fixed	1	$C = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$
SAGE-Mean[2] [17]	Message Passing	Fixed	2	$\begin{cases} C^{(1)} = I_n \\ C^{(2)} = D^{-1}A \end{cases}$
GAT[3] [28]	Self-Attention	Learnable	# of heads	$\begin{cases} \mathfrak{C}^{(s)} = A + I_n \text{ and} \\ h^{(s)}_{\boldsymbol{a}^{(l,s)}}(X^{(l)}_{i,:}, X^{(l)}_{j,:}) = \exp\big(\text{LeakyReLU}(\\ \quad (X^{(l)}_{i,:}W^{(l,s)} \,\|\, X^{(l)}_{j,:}W^{(l,s)}) \cdot \boldsymbol{a}^{(l,s)})\big) \end{cases}$

[1] Where $\widetilde{A} = A + I_n$, $\widetilde{D} = D + I_n$. [2] $C^{(2)}$ represents mean aggregator. Weight matrix in [17] is $W^{(l)} = W^{(l,1)} \| W^{(l,2)}$.
[3] Need row-wise normalization. $C^{(l,s)}_{i,j}$ is non-zero if and only if $A_{i,j} = 1$, thus GAT follows direct-neighbor aggregation.

entry, i-th row, j-th column, respectively. For a finite sequence $\langle i_b \rangle : i_1, \ldots, i_b$, we use $C_{\langle i_b \rangle,:}$ to denote the matrix whose rows are the i_b-th rows of matrix C. We use $\odot$ to denote the element-wise (Hadamard) product. $\|\cdot\|_p$ denotes the entry-wise ℓ^p norm of a vector and $\|\cdot\|_F$ denotes the Frobenius norm. We use $I_n \in \mathbb{R}^{n\times n}$ to denote the identity matrix, $\mathbf{1}_n \in \mathbb{R}^n$ to denote the vector whose entries are all ones, and $\boldsymbol{e}^i_n$ to denote the unit vector in $\mathbb{R}^n$ whose i-th entry is 1. The 0-1 indicator function is $\mathbb{1}\{\cdot\}$. We use $\mathrm{diag}(\boldsymbol{c})$ to denote a diagonal matrix whose diagonal entries are from vector $\boldsymbol{c}$. And $\|$ represents concatenation along the last axis. We use superscripts to refer to different copies of same kind of variable. For example, $X^{(l)} \in \mathbb{R}^{n\times f_l}$ denotes node representations on layer l. A Graph Neural Network (GNN) layer takes the node representation of a previous layer $X^{(l)}$ as input and produces a new representation $X^{(l+1)}$, where $X = X^{(0)}$ is the input features.

A common framework for generalized graph convolution. Although many GNNs are designed following different guiding principles including neighborhood aggregation (GraphSAGE [17], PNA [65]), spatial convolution (GCN [1]), spectral filtering (ChebNet [27], CayleyNet [66], ARMA [67]), self-attention (GAT [28], Graph Transformers [68, 69, 70]), diffusion (GDC [71], DCNN [72]), Weisfeiler-Lehman (WL) alignment (GIN [29], 3WL-GNNs [73, 74]), or other graph algorithms ([75, 76]). Despite these differences, *nearly all GNNs can be interpreted* *as performing message passing on node features, followed by feature transformation and an activation function*. As pointed out by Balcilar et al. [77], GNNs can typically be written in the form

$$X^{(l+1)} = \sigma\left(\sum_s C^{(s)} X^{(l)} W^{(l,s)}\right), \tag{3.1}$$

where $C^{(s)} \in \mathbb{R}^{n\times n}$ denotes the s-th convolution matrix that defines the message passing operator, $s \in \mathbb{Z}_+$ denotes index of convolution, and $\sigma(\cdot)$ denotes the non-linearity. $W^{(l,s)} \in \mathbb{R}^{f_l \times f_{l+1}}$ is the learnable linear weight matrix for the l-th layer and s-th filter.

Within this common framework, GNNs differ from each other by choice of convolution matrices $C^{(s)}$, which can be either fixed or learnable. A learnable convolution matrix relies on the inputs and learnable parameters and can be different in each layer (thus denoted as $C^{(l,s)}$):

$$C^{(l,s)}_{i,j} = \underbrace{\mathfrak{C}^{(s)}_{i,j}}_{\text{fixed}} \cdot \underbrace{h^{(s)}_{\theta^{(l,s)}}(X^{(l)}_{i,:}, X^{(l)}_{j,:})}_{\text{learnable}} \tag{3.2}$$

where $\mathfrak{C}^{(s)}$ denotes the fixed mask of the s-th learnable convolution, which may depend on the adjacency matrix A and input edge features $E_{i,j}$. While $h^{(s)}(\cdot,\cdot) : \mathbb{R}^{f_l} \times \mathbb{R}^{f_l} \to \mathbb{R}$ can be any learnable model parametrized by $\theta^{(l,s)}$. Sometimes a learnable convolution matrix may be further row-wise normalized as $C^{(l,s)}_{i,j} \leftarrow C^{(l,s)}_{i,j} / \sum_j C^{(l,s)}_{i,j}$, for example in GAT [28]. We stick to Eq. (3.2) in the main paper and discuss row-wise normalization in Sections 3.8.1 and 3.8.5. The receptive field of a layer of graph convolution (Eq. (3.1)) is defined as a set of nodes $\mathcal{R}^1_i$ whose features $\{X^{(l)}_{j,:} \mid j \in \mathcal{R}_i\}$ determines $X^{(l+1)}_{i,:}$. We re-formulate some popular GNNs into this generalized graph convolution framework; see Table 3.1 and Section 3.8.1 for more.

The back-propagation rule of GNNs defined by Eq. (3.1) is as follows,

$$\nabla_{X^{(l)}}\ell = \sum_s \left(C^{(l,s)}\right)^\mathsf{T} \left(\nabla_{X^{(l+1)}}\ell \odot \sigma'\left(\sigma^{-1}\left(X^{(l+1)}\right)\right)\right) \left(W^{(l,s)}\right)^\mathsf{T}, \tag{3.3}$$

which can also be understood as a form of message passing. σ' and σ^{-1} are the derivative and inverse of σ respectively and $\nabla_{X^{(l+1)}}\ell \odot \sigma'\left(\sigma^{-1}(X^{(l+1)})\right)$ is the gradients back-propagated through the non-linearity.

3.3 Scalability Problem and Theoretical Framework

When a graph is large, we are forced to mini-batch the graph by sampling a subset of $b \ll n$ nodes in each iteration. Say the node indices are $i_1, \ldots, i_b$ and a mini-batch of node features is denoted by $X_B = X_{\langle i_b \rangle,:}$. To mini-batch efficiently for any model, we hope to fetch $\theta(b)$ information to the training device, spend $\theta(Lb)$ training time per iteration while taking (n/b) iterations to traverse through the entire dataset. However, it is intrinsically difficult for most of the GNNs to meet these three scalability requirements at the same time. The receptive field of L layers of graph convolution (Eq. (3.1)) is recursively given by $\mathcal{R}_i^L = \bigcup_{j \in \mathcal{R}_i^1} \mathcal{R}_j^{L-1}$ (starting with $\mathcal{R}_i^1 \supseteq \{i\} \cup \mathcal{N}_i$), and its size grows exponentially with L. Thus, to optimize on a mini-batch of b nodes, we require $O(bd^L)$ inputs and training time per iteration. Sampling a subset of neighbors [17, 32] for each node in each layer does not change the exponential dependence on L. Although layer- [33, 34] and subgraph-sampling [30, 31] may require only $O(b)$ inputs and $O(Lb)$ training time per iteration, they are only able to consider an exponentially small proportion of messages compared with full-graph training. Most importantly, all existing sampling methods do not support dense convolution matrices with $O(n^2)$ non-zero terms. Please see Section 3.5 for a detailed comparison with sampling-based scalable methods after we introduce our framework.

Idea of dimensionality reduction. We aim to develop a scalable algorithm for any GNN models that can be re-formulated as Eq. (3.1), where the convolution matrix can be either fixed or learnable, and either sparse or dense. The major obstacle to scalability is that, for each layer of graph convolution, to compute a mini-batch of forward-passed features $X_B^{(l+1)} = X_{\langle i_b \rangle,:}^{(l+1)}$, we need $O(n)$ entries of $C_B^{(l,s)} = C_{\langle i_b \rangle,:}^{(l,s)}$ and $X^{(l)}$, which will not fit in device memory.

Our goal is to apply a dimensionality reduction to both convolution and node feature matrices, and then apply convolution using compressed "sketches" of $C_B^{(l,s)}$ and $X^{(l)}$. More specifically, we look for a projection matrix $R \in \mathbb{R}^{n \times k}$ with $k \ll n$, such that the product of low-dimensional sketches $\widetilde{C}_B^{(l,s)} = C_B^{(l,s)} R \in \mathbb{R}^{b \times k}$ and $\widetilde{X}^{(l)} = R^{\mathsf{T}} X^{(l)} \in \mathbb{R}^{k \times f_l}$ is approximately the same as $C_B^{(l,s)} X^{(l)}$. The approximated product (of all nodes) $\widetilde{C}^{(l,s)} \widetilde{X}^{(l)} = C^{(l,s)} R R^{\mathsf{T}} X^{(l)}$ can also be regarded as the result of using a low-rank approximation $C^{(l,s)} R R^{\mathsf{T}} \in \mathbb{R}^{n \times n}$ of the convolution matrix such that $\mathrm{rank}\left(C^{(l,s)} R R^{\mathsf{T}}\right) \leq k$. The distributional Johnson–Lindenstrauss lemma [78] (JL for short) shows the existence of such projection R with $m = \theta(\log(n))$, and the following result by Kane and Nelson [79] shows that R can be chosen to quite sparse:

Theorem 1. *For any convolution matrix $C \in \mathbb{R}^{n \times n}$, any column vector $X_{:,a} \in \mathbb{R}^n$ of the node feature matrix $X \in \mathbb{R}^{n \times f}$ (where $a = 1, \ldots, f$) and $\varepsilon > 0$, there exists a projection matrix $R \in \mathbb{R}^{n \times k}$ (drawn from a distribution) with only an $O(\varepsilon)$-fraction of entries non-zero, such that*

$$\Pr\left(\|C R R^{\mathsf{T}} X_{:,a} - C X_{:,a}\|_2 < \varepsilon \|C X_{:,a}\|_2\right) > 1 - \delta, \tag{3.4}$$

with $k = \theta(\log(n)/\varepsilon^2)$ and $\delta = O(1/n)$.

Now, the sketches $\widetilde{C}_B^{(l,s)}$ and $\widetilde{X}^{(l)}$ take up $O(b\log(n))$ and $\theta(f_l\log(n))$ memory respectively and can fit into the training and inference device. The sparsity of projection matrix R is favorable because:(1) if the convolution matrix $C^{(l,s)}$ is sparse (e.g., direct-neighbor message passing where only $O(d/n)$-fraction of entries are non-zero), only an $O(\varepsilon d)$-fraction of entries are non-zero in the sketch $\widetilde{C}^{(l,s)}$; (2) During training, $\widetilde{X}^{(l)}$ is updated in a "streaming" fashion using each mini-batch's inputs $X_B^{(l)}$, and a sparse R reduces the computation time by a factor of $O(\varepsilon)$. However, the projection R produced following the sparse JL-lemma [79] is randomized and requires $O(\log^2(n))$ uniform random bits to sample. It is difficult to combine this with the deterministic feed-forward and back-propagation rules of neural networks, and there is no clue when and how we should update the projection matrix. Moreover, randomized projections destroy the "identity" of each node, and for learnable convolution matrices (Eq. (3.2)), it is impossible to compute the convolution matrix only using the sketch of features $\widetilde{X}^{(l)}$. For this idea to be useful, we need a deterministic and identity-preserving construction of the projection matrix $R \in \mathbb{R}^{n\times k}$ to avoid these added complexities.

3.4 Proposed Method: Vector Quantized GNN

Dimensionality reduction using Vector Quantization (VQ). A natural and widely-used method to reduce the dimensionality of data in a deterministic and identity-preserving manner is Vector Quantization [80] (VQ), a classical data compression algorithm that can be formulated as the following optimization problem:

$$\min_{R\in\{0,1\}^{n\times k},\, \widetilde{X}\in\mathbb{R}^{k\times f}} \|X - R\widetilde{X}\|_F \quad \text{s.t. } R_{i,:} \in \{\boldsymbol{e}_k^1, \ldots, \boldsymbol{e}_k^k\}, \tag{3.5}$$

which is classically solved via *k-means* [80]. Here the sketch of features $\widetilde{X}$ is called the feature "codewords." R is called the codeword assignment matrix, whose rows are unit vectors in $\mathbb{R}^k$, i.e., $R_{i,v} = 1$ if and only if the i-th node is assigned to the v-th cluster in *k-means*. The objective in Eq. (3.5) is called Within-Cluster Sum of Squares (WCSS), and we can define the relative error of VQ as $\epsilon = \|X - R\widetilde{X}\|_F / \|X\|_F$. The rows of $\widetilde{X}$ are the k codewords (i.e., centroids in *k-means*), and can be computed as $\widetilde{X} = \mathrm{diag}^{-1}(R^\mathsf{T}\mathbf{1}_n)R^\mathsf{T}X$, which is slightly different from the definition in Section 3.3 as a row-wise normalization of R^T is required. The sketch of the convolution matrix $\widetilde{C}$ can still be computed as $\widetilde{C} = CR$. In general, VQ provides us a principled framework to learn the low-dimensional sketches $\widetilde{X}$ and $\widetilde{C}$, in a deterministic and node-identity-preserving manner. However, to enable mini-batch training and inference of GNNs using VQ, three more questions need to be answered:

- How to approximate the forward-passed mini-batch features of nodes using the learned codewords?
- How to back-propagate through VQ and estimate the mini-batch gradients of nodes?
- How to update the codewords and assignment matrix along with the training of GNN?

In the following part of this section, we introduce the VQ-GNN algorithm by answering all the three questions and presenting a scalability analysis.

Approximated forward and backward message passing. To approximate the forward pass through a GNN layer (Eq. (3.1)) with a mini-batch of nodes $\langle i_b \rangle$, we can divide the messages into two categories: intra-mini-batch messages, and messages from out-of-mini-batch nodes; see the right figure of Fig. 3.1. Intra-mini-batch messages $C_{\text{in}}^{(l,s)} X_B^{(l)}$ can always be computed exactly, where $C_{\text{in}}^{(l,s)} = (C_B^{(l,s)})_{:,\langle i_b \rangle} \in \mathbb{R}^{b \times b}$, because they only rely on the previous layer's node features of the current mini-batch. Equipped with the codewords $\widetilde{X}^{(l)}$ and the codeword assignment of all nodes $R^{(l)}$, we can approximate the messages from out-of-mini-batch nodes as $\widetilde{C}_{\text{out}}^{(l,s)}\widetilde{X}^{(l)}$, where $\widetilde{X}^{(l)} = \text{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)}$ as defined above and $\widetilde{C}_{\text{out}}^{(l,s)} = C_{\text{out}}^{(l,s)}R$. Here, $C_{\text{out}}^{(l,s)}$ is the remaining part of the convolution matrix after removing the intra-mini-batch messages, thus $(C_{\text{out}}^{(l,s)})_{:,j} = (C_B^{(l,s)})_{:,j}\mathbb{1}\{j \in \langle i_b\rangle\}$ for any $j \in \{1,\ldots,n\}$, and $\widetilde{C}^{(l,s)}$ is the sketch of $C_{\text{out}}^{(l,s)}$. In general, we can easily approximate the forward-passed mini-batch features $X_B^{(l+1)}$ by $\widehat{X}_B^{(l+1)} = \sigma\big(\sum_s (C_{\text{in}}^{(l,s)}X_B^{(l)} + \widetilde{C}_{\text{out}}^{(l,s)}\widetilde{X}^{(l)})W^{(l,s)}\big)$.

However, the above construction of $\widehat{X}_B^{(l+1)}$ does not allow us to back-propagate through VQ straightforwardly using chain rules. During back-propagation, we aim at approximating the previous layer's mini-batch gradients $\nabla_{X_B^{(l)}}\ell$ given the gradients of the (approximated) output $\nabla_{\widehat{X}_B^{(l+1)}}\ell$ (Eq. (3.3)). Firstly, we do not know how to compute the partial derivative of $\widetilde{C}_{\text{out}}^{(l,s)}$ and $\widetilde{X}^{(l)}$ with respect to $X_B^{(l)}$, because the learning and updating of VQ codewords and assignment are *data dependent* and are usually realized by an iterative optimization algorithm. Thus, we need to go through an iterative computation graph to evaluate the partial derivative of $R^{(l)}$ with respect to $X_B^{(l)}$, which requires access to many historical features and gradients, thus violating the scalability constraints. Secondly, even if we apply some very rough approximation during back-propagation as in [81], that is, assuming that the partial derivative of $R^{(l)}$ with respect to $X_B^{(l)}$ can be ignored (i.e., the codeword assignment matrix is detached from the computation graph, known as "straight through" back-propagation), we are not able to evaluate the derivatives of codewords $\widetilde{X}^{(l)}$ because they rely on some node features out of the current mini-batch and are not in the training device. Generally speaking, designing a back-propagation rule for VQ under the mini-batch training setup is a challenging new problem.

It is helpful to re-examine what is happening when we back-propagate on the full graph.

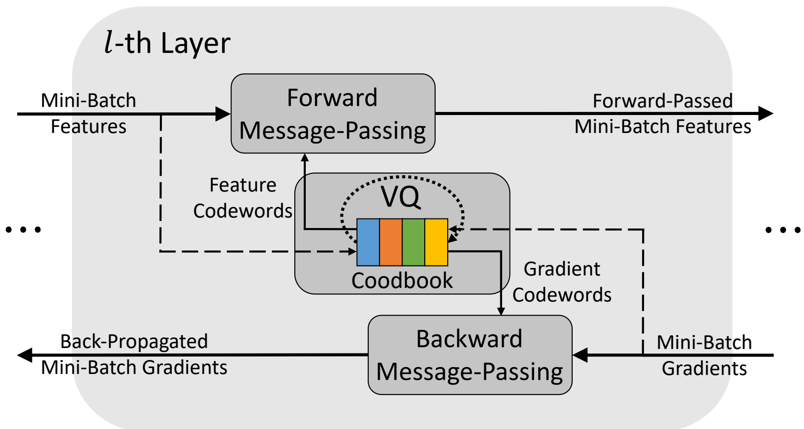


Figure 3.2: Three types of messages contribute to the mini-batch features and gradients. We only need "red" and "green" messages for the forward-pass. However, "blue" messages are required for back-propagation. The "red", "blue", and "green" messages are characterized by $\widetilde{C}_{\mathrm{out}}$, $(\widetilde{C^{\mathsf{T}}})_{\mathrm{out}}$, and C_{in} respectively (Eqs. (3.6) and (3.7)).

In Section 3.2, we see that back-propagation of a layer of convolution-based GNN can also be realized by message passing (Eq. (3.3)). In Fig. 3.2, we show the messages related to a mini-batch of nodes can be classified into three types. The "green" and "red" messages are the intra-mini-batch messages and the messages from out-of-mini-batch nodes, respectively. Apart from them, although the "blue" messages to out-of-mini-batch nodes do not contribute to the forward-passed mini-batch features, they are used during back-propagation and are an important part of the back-propagated mini-batch gradients. Since both forward-pass and back-propagation can be realized by message passing, can we approximate the back-propagated mini-batch gradients $\nabla_{X_B^{(l)}}\ell$ in a symmetric manner? We can introduce a set of gradient codewords $\widetilde{G}^{(l+1)} = \mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}G^{(l+1)}$ using the same assignment matrix, where $G^{(l+1)} = \nabla_{\widehat{X}^{(l+1)}}\ell \odot \sigma'\big(\sigma^{-1}(X^{(l+1)})\big)$ is the gradients back-propagated through non-linearity. Each gradient codeword corresponds one-to-one with a feature codeword since we want to use only one assignment matrix R. Each pair of codewords are concatenated together during VQ updates. Following this idea, we define the approximated

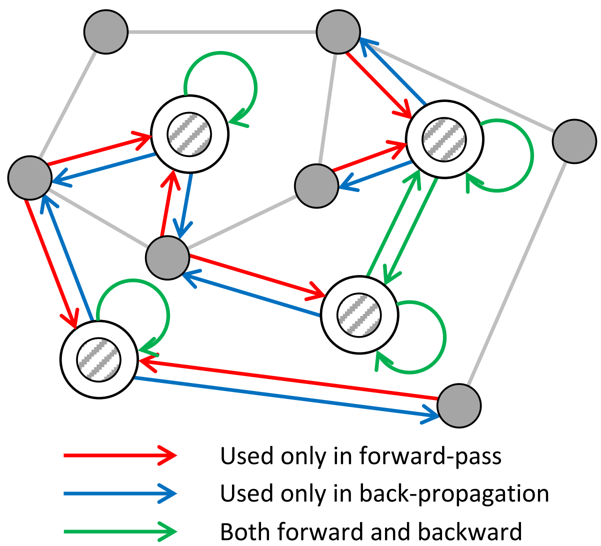


Figure 3.3: For each layer, VQ-GNN estimates the forward-passed mini-batch features using the previous layer's mini-batch features and the feature codewords through approximated forward message-passing (Eq. (3.6)). The back-propagated mini-batch gradients are estimated in a symmetric manner with the help of gradient codewords (Eq. (3.7)).

forward and backward message passing as follows:

$$\begin{bmatrix}\widehat{X}_B^{(l+1)}\\ \cdot\end{bmatrix} = \sigma\left(\sum_s \underbrace{\begin{bmatrix} C_{\text{in}}^{(l,s)} & \widetilde{C}_{\text{out}}^{(l,s)} \\ (\widetilde{C^{(l,s)\mathsf{T}}})_{\text{out}} & \mathbf{0}\end{bmatrix}}_{\substack{\text{approx. message passing}\\ \text{weight matrix } \mathscr{C}^{(l,s)}}} \underbrace{\begin{bmatrix} X_B^{(l)} \\ \widetilde{X}^{(l)}\end{bmatrix}}_{\substack{\text{mini-batch features}\\ \text{and feat. codewords}}} W^{(l,s)}\right), \tag{3.6}$$

$$\begin{bmatrix}\widehat{\nabla}_{X_B^{(l)}}\ell\\ \cdot\end{bmatrix} = \sum_s \left(\mathscr{C}^{(l,s)}\right)^{\mathsf{T}} \underbrace{\begin{bmatrix} G_B^{(l+1)} \\ \widetilde{G}^{(l+1)}\end{bmatrix}}_{\substack{\text{mini-batch gradients}\\ \text{and grad. codewords}}} \left(W^{(l,s)}\right)^{\mathsf{T}}, \tag{3.7}$$

where $\mathscr{C}^{(l,s)} \in \mathbb{R}^{(b+m)\times(b+m)}$ is the approximated message passing weight matrix and is shared during the forward-pass and back-propagation process. The lower halves of the left-hand side vectors of Eqs. (3.6) and (3.7) are used in neither the forward nor the backward calculations and are never calculated during training or inference. The approximated forward and backward message passing enables the end-to-end mini-batch training and inference of GNNs and is the core of our VQ-GNN framework.

Error-bounds on estimated features and gradients. We can effectively upper bound the estimation errors of mini-batch features and gradients using the relative error ϵ of VQ under some mild conditions. For ease of presentation, we assume the GNN has only one convolution matrix in the following theorems.

Theorem 2. *If the VQ relative error of l-th layer is $\epsilon^{(l)}$, the convolution matrix $C^{(l)}$ is either fixed or learnable with the Lipschitz constant of $h_{\theta^{(l)}}(\cdot):\mathbb{R}^{2f_l}\to\mathbb{R}$ upper-bounded by $\mathcal{L}ip(h_{\theta^{(l)}})$, and the Lipschitz constant of the non-linearity is $\mathcal{L}ip(\sigma)$, then the estimation error of forward-passed mini-batch features satisfies,*

$$\|\widehat{X}_B^{(l+1)} - X_B^{(l+1)}\|_F \leq \epsilon^{(l)} \cdot (1 + O(\mathcal{L}ip(h_{\theta^{(l)}})))\mathcal{L}ip(\sigma)\|C^{(l)}\|_F\|X^{(l)}\|_F\|W^{(l)}\|_F. \tag{3.8}$$

Corollary 3. *If the conditions in Theorem 2 hold and the non-linearity satisfies $|\sigma'(z)| \leq \sigma'_{\max}$ for any $z\in\mathbb{R}$, then the estimation error of back-propagated mini-batch gradients satisfies,*

$$\|\widehat{\nabla}_{X_B^{(l)}}\ell - \nabla_{X_B^{(l)}}\ell\|_F \leq \epsilon^{(l)} \cdot (1 + O(\mathcal{L}ip(h_{\theta^{(l)}}))\sigma'_{\max}\|C^{(l)}\|_F\|\nabla_{X^{(l+1)}}\ell\|_F\|W^{(l)}\|_F. \tag{3.9}$$

Note that the error bounds rely on the Lipschitz constant of $h(\cdot)$ when the convolution matrix is learnable. In practice, we can Lipshitz regularize GNNs like GAT [28] without affecting their performance; see Section 3.8.5.

VQ-GNN: the complete algorithm and analysis of scalability. The only remaining question is how to update the learned codewords and assignments during training? In this paper, we use the VQ update rule proposed in [81], which updates the codewords as exponential moving averages of the mSeini-batch inputs; see Section 3.8.5 for the detailed algorithm. We find such an exponential moving average technique suits us well for the mini-batch training of GNNs and resembles the *online k-means* algorithm. See Fig. 3.3 for the schematic diagram of VQ-GNN, and the complete pseudo-code is in Section 3.8.5.

With VQ-GNN, we can mini-batch train and perform inference on large graphs using GNNs, just like a regular neural network (e.g., MLP). We have to maintain a small codebook of k codewords and update it for each iteration, which takes an extra $O(Lkf)$ memory and $O(Lnkf)$ training time per epoch, where L and f are the numbers of layers and (hidden) features of the GNN respectively. We can effectively preserve all messages related to a mini-batch while randomly sampling nodes from the graph. The number of intra-mini-batch messages is $O(b^2d/n)$ when the nodes are sampled randomly. Thus we only need to pass $O(b^2d/n + bk)$ messages per iteration and $O(bd + nk)$ per epoch. In practice, when combined with techniques including product VQ and implicit whitening (see Section 3.8.5), we can further improve the stability and performance of VQ-GNN. These theoretical and experimental analyses justify the efficiency of the proposed VQ-GNN framework.

3.5 Related Work

In this section, we review some of the recent scalable GNN methods and analyze their theoretical memory and time complexities, with a focus on scalable algorithms that can be

Table 3.2: Memory and time complexities of sampling-based methods and our approach; see Section 3.5 for details.

Scalable Method	Memory Usage	Pre-computation Time	Training Time	Inference Time
NS-SAGE	$O(br^L f + Lf^2)$	—	$O(nr^L f + nr^{L-1} f^2)$	
Cluster-GCN	$O(Lbf + Lf^2)$	$O(m)$	$O(Lmf + Lnf^2)$	$O(nd^L f + nd^{L-1} f^2)$
GraphSAINT-RW	$O(L^2 bf + Lf^2)$	—	$O(L^2 nf + L^2 nf^2)$	
VQ-GNN (Ours)	$O(Lbf + Lf^2 + Lkf)$	—	$O(Lbdf + Lnf^2 + Lnkf)$	$O(Lbdf + Lnf^2)$

universally applied to a variety of GNN models (like our VQ-GNN framework), including NS-SAGE[1] [17], Cluster-GCN [30], and GraphSAINT [31]. We consider GCN here as the simplest benchmark. For a GCN with L layers and f-dimensional (hidden) features in each layer, when applied to a sparse graph with n nodes and m edges (i.e., average degree $d = m/n$) for "full-graph" training and inference: the memory usage is $O(Lnf + Lf^2)$ and the training/inference time is $O(Lmf + Lnf^2)$. We further assume the graph is large and consider the training and inference device memory is $O(b)$ where b is the mini-batch size (i.e., the memory bottleneck limits the mini-batch size), and generally $d \ll b \ll n \ll m$ holds. We divide sampling baselines into three categories, and the complexities of selected methods are in Table 3.2. See Section 3.8.4 for more related work discussions.

Neighbor-sampling. Neighbor sampling scheme chooses a subset of neighbors in each layer to reduce the amount of message passing required. NS-SAGE [17] samples r neighbors for each node and only aggregates the messages from the sampled node. For a GNN with L layers, $O(br^L)$ nodes are sampled in a mini-batch, which leads to the complexities growing exponentially with the number of layers L; see Table 3.2. Therefore, NS-SAGE is not scalable on large graphs for a model

[1]We call the neighbor sampling method in [17] NS-SAGE and the GNN model in the same paper SAGE-Mean to avoid ambiguity.

with an arbitrary number of layers. NS-SAGE requires all the neighbors to draw non-stochastic predictions in the inference phase, resulting in a $O(d^L)$ inference time since we cannot fit $O(n)$ nodes all at once to the device. VR-GCN [32] proposes a variance reduction technique to further reduce the size r of sampled neighbors. However, VR-GCN requires a $O(Lnf)$ side memory of all the nodes' hidden features and suffers from this added memory complexity.

Layer-sampling. These methods perform node sampling independently in each layer, which results in a constant sample size across all layers and limits the exponential expansion of neighbor size. FastGCN [33] applies importance sampling to reduce variance. Adapt [34] improves FastGCN by an additional sampling network but also incurs the significant overhead of the sampling algorithm.

Subgraph-sampling. Some proposed schemes sample a subgraph for each mini-batch and perform forward and backward passes on the same subgraph across all layers. Cluster-GCN [30] partitions a large graph into several densely connected subgraphs and samples a subset of subgraphs (with edges between clusters added back) to train in each mini-batch. Cluster-GCN requires $O(m)$ pre-computation time and $O(bd)$ time to recover the intra-cluster edges when loading each mini-batch. GraphSAINT [31] samples a set of nodes and takes the induced subgraph for mini-batch training. We consider the best-performing variant, GraphSAINT-RW, which uses L steps of random walk to induce subgraph from b randomly sampled nodes. $O(Lb)$ nodes and edges are covered in each of the n/b mini-batches. Although $O(Ln)$ nodes are sampled with some repetition in an epoch, the number of edges covered (i.e., messages considered in each layer of a GNN) is also $O(Ln)$ and is usually much smaller than m. GraphSAINT-Node, which randomly samples nodes for

Table 3.3: Peak memory usage. Evaluated when fixing the number of gradient-descended nodes or the number of messages passed per mini-batch to be the same. Tested for GCN and SAGE-Mean on the *ogbn-arxiv* benchmark.

Fixed GNN Model	85K nodes per batch GCN	85K nodes per batch SAGE-Mean	1.5M messages passed per batch GCN	1.5M messages passed per batch SAGE-Mean
NS-SAGE	—	1140.3 MB	—	953.7 MB
Cluster-GCN	501.5 MB	514.1 MB	757.4 MB	769.3 MB
GraphSAINT-RW	526.5 MB	519.2 MB	661.6 MB	650.4 MB
VQ-GNN (Ours)	758.0 MB	801.8 MB	485.5 MB	508.5 MB

each mini-batch, does not suffer from this L factor in the complexities. However, its performance is worse than GraphSAINT-RW's. Like NS-SAGE and some other sampling methods, Cluster-GCN and GraphSAINT-RW cannot draw predictions on a randomly sampled subgraph in the inference phase. Thus they suffer from the same $O(d^L)$ inference time complexity as NS-SAGE; see Table 3.2.

3.6 Experiments

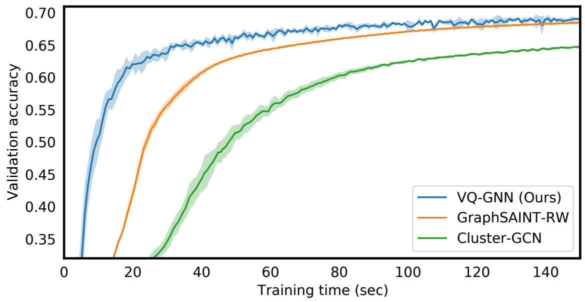
In this section, we verify the efficiency, robustness, and universality of VQ-GNN using a series of experiments. See Section 3.8.6 for implementation details and Section 3.8.7 for ablation studies and more experiments.

Scalability and efficiency: memory usage, convergence, training and inference time. We summarize the *memory usage* of scalable methods and our VQ-GNN framework in Table 3.3. Based on the implementations of the PyG library [82], memory consumption of GNN models usually grows linearly with respect to both the number of nodes and the number of edges in a mini-batch. On the *ogbn-arxiv* benchmark, we fix the number of gradient-descended nodes and the number of messages passed in a mini-batch to be 85K and 1.5M respectively for fair comparisons among the sampling methods and our approach. VQ-GNN might require some small extra memory when provided with the same amount of nodes per batch, which is the cost to retain all the edges from the original graph. However, our VQ-GNN framework can effectively preserve all the edges connected to a mini-batch of nodes (i.e., never drop edges); see Fig. 3.1. Thus when we fix the number of messages passed per batch, our method can show significant memory efficiency compared with the sampling baselines.

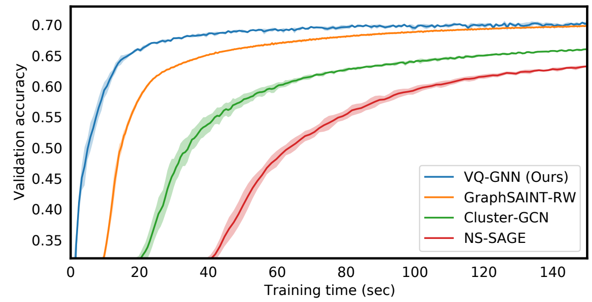
Fig. 3.4 shows the convergence comparison of various scalability methods, where we see VQ-GNN is superior in terms of the *convergence speed* with respect to the *training time*. When training GCN and SAGE-Mean on the *ogbn-arxiv* benchmark for a specific amount of time (e.g., $100\ \mathrm{s}$), the validation performance of VQ-GNN is always the highest. The training time in Fig. 3.4 excludes the time for data loading, pre-processing, and validation set evaluation.

Our VQ-GNN approach also leads to compelling *inference* speed-ups. Despite the training-efficiency issues of GNNs, conducting inference on large-scale graphs suffers from some unique challenges. According to our discussions in Section 3.5, and following the standard implementations provided by the Open Graph Benchmark (OGB) [12], the three sampling-based baselines (which share the same inference procedure) require all of the L-hop neighbors of the mini-batch nodes to lie on the device at once during the inference phase. The inference time of SAGE-Mean trained with sampling-methods on the *ogbn-arxiv* benchmark is $1.61\ \mathrm{s}$, while our method can accelerate inference by an order of magnitude and reduce the inference time to $0.40\ \mathrm{s}$.

Performance comparison across various datasets, settings, and tasks. We validate the efficacy of our method on various benchmarks in Table 3.4. The four representative benchmarks are selected because they have very different types of datasets, settings, and tasks. The *ogbn-arxiv* benchmark



((a)) GCN



((b)) SAGE-Mean

Figure 3.4: Convergence curves (validation accuracy vs. training time). Mini-batch size and learning rate are kept the same. Tested for GCN and SAGE-Mean on the *ogbn-arxiv* benchmark.

is a common citation network of arXiv papers, while *Reddit* is a very dense social network of Reddit posts, which has much more features per node and larger average node degree; see Table 3.6 in Section 3.8.6 for detailed statistics of datasets. *PPI* is a node classification benchmark under the inductive learning setting, i.e., neither attributes nor connections of test nodes are present during training, while the other benchmarks are all transductive. VQ-GNN can be applied under the inductive setting with only one extra step: during the inference stage, we now need to find the codeword assignments (i.e., the nearest codeword) of the test nodes before making predictions since we have no access to the test nodes during training. Neither the learned codewords nor the GNN parameters are updated during inference. *ogbl-collab* is a link prediction benchmark where the labels and loss are intrinsically different.

It is very challenging for a scalable method to perform well on all benchmarks. In Table 3.4, we confirm that VQ-GNN is more *robust* than the three sampling-based methods. Across the four benchmarks, VQ-GNN can always achieve performance similar with or better than the oracle "full-graph" training performance, while the other scalable algorithms may suffer from performance drop in some cases. For example, NS-SAGE fails when training GAT on *ogbl-collab*, Cluster-GCN consistently falls behind on *PPI*, and GraphSAINT-RW's performance drops on the *ogbl-collab*

Table 3.4: Performance comparison between sampling-based baselines and our approach, VQ-GNN.

Task Benchmark	Node Classification (Transductive) *ogbn-arxiv* (Acc.±std.)			Node Classification (Transductive) *Reddit* (Acc.±std.)		
GNN Model	GCN	SAGE-Mean	GAT	GCN	SAGE-Mean	GAT
"Full-Graph"	.7029 ± .0036	.6982 ± .0038	.7097 ± .0035	OOM[2]	OOM[2]	OOM[2]
NS-SAGE.	NA[1]	.7094 ± .0060	.7123 ± .0044	NA[1]	.9615 ± .0089	.9426 ± .0043
Cluster-GCN	.6805 ± .0074	.6976 ± .0049	.6960 ± .0062	.9264 ± .0034	.9456 ± .0061	.9380 ± .0055
GraphSAINT-RW	.7079 ± .0057	.6987 ± .0039	.7117 ± .0032	.9225 ± .0057	.9581 ± .0074	.9431 ± .0067
VQ-GNN (Ours)	.7055 ± .0033	.7028 ± .0047	.7043 ± .0034	.9399 ± .0021	.9449 ± .0024	.9438 ± .0059
Task Benchmark	**Node Classification (Inductive) *PPI* (F1-score[3]±std.)**			**Link Prediction (Transductive) *ogbl-collab* (Hits@50±std.)**		
GNN Model	GCN	SAGE-Mean	GAT	GCN	SAGE-Mean	GAT
"Full-Graph"	.9173 ± .0039	.9358 ± .0046	.9722 ± .0035	.4475 ± .0107	.4810 ± .0081	.4048 ± .0125
NS-SAGE.	NA[1]	.9121 ± .0033	.9407 ± .0025	NA[1]	.4776 ± .0041	.3499 ± .0142
Cluster-GCN	.8852 ± .0066	.8810 ± .0091	.9051 ± .0077	.4068 ± .0096	.3486 ± .0216	.3905 ± .0152
GraphSAINT-RW	.9110 ± .0057	.9382 ± .0074	.9612 ± .0042	.4368 ± .0169	.3359 ± .0128	.3489 ± .0114
VQ-GNN (Ours)	.9549 ± .0058	.9578 ± .0019	.9737 ± .0033	.4316 ± .0134	.4673 ± .0164	.4102 ± .0099

[1] NS-SAGE sampling method is not compatible with the GCN backbone. [2] "OOM" refers to "out of memory". The "full-graph" training on the *Reddit* benchmark requires more than 11 `GB` of memory. [3] The *PPI* benchmark comes with multiple labels per node, and the evaluation metric is F1 score instead of accuracy.

when using SAGE-Mean and GAT backbones. We think the robust performance of VQ-GNN is its unique value among the many other scalable solutions. VQ-GNN framework is robust because it provides bounded approximations of "full-graph" training (Theorem 2 and Corollary 3), while most of the other scalable algorithms do not enjoy such a theoretical guarantee. VQ-GNN is also *universal* to various backbone models, including but not limited to GCN, SAGE-Mean, and GAT shown here; see Section 3.8.7 for more experiments on GNNs that utilize multi-hop neighborhoods and global context, e.g., graph transformers.

3.7 Conclusion

Summary of our framework: strengths, weaknesses, future directions, and broader impacts. This paper introduced the proposed VQ-GNN framework, which can scale most state-of-the-art

GNNs to large graphs through a principled and fundamentally different approach compared with sampling-based methods. We have shown both theoretically and experimentally that our approach is efficient in memory usage, training and inference time, and convergence speed. VQ-GNN can be universally applied to most GNN models and different graph learning tasks and can equivalently scale-up GNNs utilizing many-hops-away or global context for each layer. However, the performance of VQ-GNN relies on the quality of approximation provided by VQ. In practice, for VQ to work adequately in GNN, a set of techniques are necessary. Because of the limited time, we did not heuristically explore all possible techniques or optimize the VQ design. Given that our preliminary design of VQ in GNN already achieved competitive performance compared with the state-of-the-art sampling baselines, we hypothesize that further optimization of VQ design could improve performance. We hope our work opens up promising new avenues of research for scaling up GNNs, which also has the potential to be applied to other data domains wherever the size of a single sample is large, e.g., long time-series or videos. Considering broader impacts, we view our work mainly as a methodological and theoretical contribution, which paves the way for more resource-efficient graph representation learning. We envision our methodological innovations can enable more scalable ways to do large-network analysis for social good. However, progress in graph embedding learning might also trigger other hostile social network analyses, e.g., extracting fine-grained user interactions for social tracking.

3.8 Appendix

3.8.1 Generalized Graph Convolution Framework

In this section, we present more results and discussions regarding the common generalized graph convolution framework in Section 3.2.

Summary of GNNs re-formulated into the common graph convolution framework. As stated in Section 3.2, most GNNs can be interpreted as performing message passing on node features, followed by feature transformation and an activation function (Eq. (3.6)), which is known as the common "generalized graph convolution" framework. We list more GNN models that fall into this framework in the following Table 3.5.

Table 3.5: Summary of more GNNs re-formulated as generalized graph convolution.

Model Name	Design Idea	Conv. Matrix Type	# of Conv.	Convolution Matrix
GIN[1] [29]	WL-Test	Fixed + Learnable	2	$\begin{cases} C^{(1)} = A \\ \mathfrak{C}^{(2)} = I_n \text{ and } h^{(2)}_{\epsilon^{(l)}} = 1 + \epsilon^{(l)} \end{cases}$
ChebNet[2] [27]	Spectral Conv.	Learnable	order of poly.	$\begin{cases} \mathfrak{C}^{(1)} = I_n, \mathfrak{C}^{(2)} = 2L/\lambda_{\max} - I_n, \\ \mathfrak{C}^{(s)} = 2\mathfrak{C}^{(2)}\mathfrak{C}^{(s-1)} - \mathfrak{C}^{(s-2)} \\ \text{and } h^{(s)}_{\theta^{(s)}} = \theta^{(s)} \end{cases}$
GDC[3] [71]	Diffusion	Fixed	1	$C = S$
Graph Transformers[4] [68, 69, 70]	Self-Attention	Learnable	# of heads	$\begin{cases} \mathfrak{C}^{(s)}_{i,j} = 1 \text{ and } h^{(s)}_{(W_Q^{(l,s)}, W_K^{(l,s)})}(X^{(l)}_{i,:}, X^{(l)}_{j,:}) \\ = \exp\left(\frac{1}{\sqrt{d_{k,l}}}(X^{(l)}_{i,:} W_Q^{(l,s)})(X^{(l)}_{j,:} W_K^{(l,s)})^\top\right) \end{cases}$

[1] The weight matrices of the two convolution supports are the same, $W^{(l,1)} = W^{(l,2)}$. [2] Where normalized Laplacian $L = I_n - D^{-1/2}AD^{-1/2}$ and $\lambda_{\max}$ is its largest eigenvalue, which can be approximated as 2 for a large graph. [3] Where S is the diffusion matrix $S = \sum_{k=0}^{\infty} \theta_k \boldsymbol{T}^k$, for example, decaying weights $\theta_k = e^{-t}\frac{t^k}{k!}$ and transition matrix $\boldsymbol{T} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$. [4] Need row-wise normalization. Only describes the global self-attention layer, where $W_Q^{(l,s)}, W_Q^{(l,s)} \in \mathbb{R}^{f_l, d_{k,l}}$ are weight matrices which compute the queries and keys vectors. In contrast to GAT, all entries of $\mathfrak{C}^{(l,s)}_{i,j}$ are non-zero. Different design of Graph Transformers [68, 69, 70] use graph adjacency information in different ways, and is not characterized here, see the original papers for details.

Receptive fields. The model in the top part of Table 3.5 (GIN) and the models in Table 3.1 in Section 3.2 (GCN, SAGE-Mean, and GAT) follow direct-neighbor message passing, and their single-layer receptive fields, defined as a set of nodes $\mathcal{R}_i^1$ whose features $\{X_{j,:}^{(l)} \mid j \in \mathcal{R}_i\}$ determines $X_{i,:}^{(l+1)}$, are exactly $\mathcal{R}_i^1 = \{i\} \cup \mathcal{N}_i$, where $\mathcal{N}_i$ is the set of direct neighbors of node i. The models in the bottom part of Table 3.5 (ChebNet, GDC, Transformer) can utilize many-hops-away or gloabl context each layer, and their single-layer receptive field $\mathcal{R}_i^1 \supseteq \{i\} \cup \mathcal{N}_i$.

GNNs that cannot be defined as graph convolution. Some GNNs, including Gated Graph Neural Networks [83] and ARMA Spectral Convolution Networks [67] cannot be re-formulated into this common graph convolution framework because they rely on either Recurrent Neural Networks (RNNs) or some iterative processes, which are out of the paradigm of message passing.

Further row-wise normalization of a learnable convolution matrix. Sometimes a learnable convolution matrix many be further row-wise normalized as $C_{i,j}^{(l,s)} \leftarrow C_{i,j}^{(l,s)} / \sum_j C_{i,j}^{(l,s)}$. This is required for self-attention-based GNNs, including GAT and Graph Transformers. This normalization process will not affect the single-layer receptive fields of the two models and can be handled by a small modification to the VQ-GNN algorithm; see Section 3.8.5.

3.8.2 Proofs of Theoretical Results

This section provides the formal proofs of the theoretical results in the paper.

Proof of Theorem 1. The proof is based on the sparse distributional Johnson–Lindenstrauss (JL) lemma in [79], which is stated as the following Lemma 1 for completeness.

Lemma 1. *For any integer $n > 0$, and any $0 < \varepsilon$, $\delta < 1/2$, there exists a probability distribution of $R \in \mathbb{R}^{n\times k}$ where R has only an $O(\varepsilon)$-fraction of non-zero entries, such that for any $x \in \mathbb{R}^d$,*

$$\Pr\left((1-\varepsilon)\|x\|_2 \leq \|R^\mathsf{T} x\|_2 \leq (1+\varepsilon)\|x\|_2\right) > 1-\delta', \tag{3.10}$$

with $k = \theta(\varepsilon^{-2}\log(1/\delta))$ and thus $\delta' = O(\mathrm{e}^{-\varepsilon^2 k})$.

Proof of Theorem 1: From the given lemma, consider the inner product $x^\mathsf{T} y$ between any two vectors $x, y \in \mathbb{R}^n$. Using the union-bound, one can derive that,

$$\Pr\left(|x^\mathsf{T} RR^\mathsf{T} y - x^\mathsf{T} y| \leq \varepsilon |x^\mathsf{T} y|\right) > 1-2\delta'.$$

Now, let x be any row vector of C and y be any column vector of X, i.e., $x = C_{b,:}$ and $y = X_{:,a}$ for any $b \in \{1,\ldots,n\}$ and $a \in \{1,\ldots,f\}$, then we get,

$$\Pr\left(|C_{b,:}RR^\mathsf{T} X_{:,a} - C_{b,:}X_{:,a}| \leq \varepsilon |C_{b,:}X_{:,a}|\right) > 1-2\delta'.$$

Therefore, by the union bound again, we have,

$$\begin{aligned}&\Pr\left(\|CRR^\mathsf{T} X_{:,a} - CX_{:,a}\|_2 \leq \varepsilon \|CX_{:,a}\|_2\right)\\ &\qquad \geq 1 - \sum_{b=1}^{n} \Pr\left(|C_{b,:}RR^\mathsf{T} X_{:,a} - C_{b,:}X_{:,a}| > \varepsilon |C_{b,:}X_{:,a}|\right) > 1-2n\delta'.\end{aligned}$$

Now we denote $\delta = 2n\delta'$. The required $\delta = O(1/n)$ can be achieved when $k =$

$\theta(\log(n)/\varepsilon^2)$. Thus we finally obtain,

$$\Pr\left(\|CRR^{\mathsf{T}}X_{:,a} - CX_{:,a}\|_2 < \varepsilon\|CX_{:,a}\|_2\right) > 1 - \delta,$$

with $k = \theta(\log(n)/\varepsilon^2)$ and $\delta = O(1/n)$, which concludes the proof. $\square$

Proof of Theorem 2. The proof is a direct application of the VQ relative-error bound and the Lipschitz continuity properties.

Proof of Theorem 2: If we denote the learnable convolution matrix calculated using the approximated features (i.e., feature codewords) by $C'^{(l)}$, then we have,

$$\begin{aligned}\|\widehat{X}_B^{(l+1)} - X_B^{(l+1)}\|_F &\leq \mathcal{L}ip(\sigma)\|C'^{(l)}R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)}W^{(l)} - C^{(l)}X^{(l)}W^{(l)}\|_F \\ &\leq \mathcal{L}ip(\sigma)\|C'^{(l)}R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)} - C^{(l)}X^{(l)}\|_F\|W^{(l)}\|_F,\end{aligned}$$

where

$$\begin{aligned}&\|C'^{(l)}R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)} - C^{(l)}X^{(l)}\|_F \\ &\quad\leq \|C'^{(l)} - C^{(l)}\|_F\|R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)}\|_F \\ &\qquad + \|C^{(l)}\|_F\|X^{(l)} - R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)}\|_F.\end{aligned} \tag{3.11}$$

The relative error $\epsilon^{(l)}$ of VQ of the l-th layer is defined as,

$$\|X^{(l)} - R\,\mathrm{diag}^{-1}(R^{\mathsf{T}}\mathbf{1}_n)R^{\mathsf{T}}X^{(l)}\|_F \leq \epsilon^{(l)}\|X^{(l)}\|_F.$$

Thus the second term on the right-hand side of Eq. (3.11) satisfies,

$$\|C^{(l)}\|_F \|X^{(l)} - R\,\mathrm{diag}^{-1}(R^\mathsf{T}\mathbf{1}_n)R^\mathsf{T}X^{(l)}\|_F \leq \epsilon^{(l)} \cdot \|C^{(l)}\|_F\|X^{(l)}\|_F$$

For the first term on the right-hand side of Eq. (3.11) satisfies, first note that,

$$\|R\,\mathrm{diag}^{-1}(R^\mathsf{T}\mathbf{1}_n)R^\mathsf{T}\|_F = \sqrt{k}$$

is a constant where k is the number of codewords.

And $C'^{(l)}$ is different to $C^{(l)}$, because the convolution matrix is learnable. We have

$$C^{(l)}_{i,j} = \mathfrak{C}_{i,j}h_{\theta^{(l)}}(X_{i,:}, X_{j,:}) \quad \text{and} \quad C'^{(l)}_{i,j} = \mathfrak{C}_{i,j}h_{\theta^{(l)}}(\widetilde{X}^{(l)}_{i,:}, \widetilde{X}^{(l)}_{j,:}),$$

where $\widetilde{X}^{(l)} = \mathrm{diag}^{-1}(R^\mathsf{T}\mathbf{1}_n)R^\mathsf{T}X^{(l)}$.

Therefore, using the Lipschitz constant of $h_{\theta^{(l)}}$, because for any $i \in \{1, \ldots, n\}$,

$$\|X_{i,:} - \widetilde{X}^{(l)}_{i,:}\|_2 \leq \|X^{(l)} - R\,\mathrm{diag}^{-1}(R^\mathsf{T}\mathbf{1}_n)R^\mathsf{T}X^{(l)}\|_F \leq \epsilon^{(l)}\|X^{(l)}\|_F,$$

we have for any $i, j \in \{1, \ldots, n\}$,

$$|C'^{(l)}_{i,j} - C^{(l)}_{i,j}| \leq 2|\mathfrak{C}_{i,j}| \cdot \mathcal{L}ip(h_{\theta^{(l)}})\epsilon^{(l)}\|X^{(l)}\|_F$$

Summing up for all $(i, j) \in \{1, \ldots, n\}^2$, we have, for the first term on the right-hand side of Eq. (3.11),

$$\|C'^{(l)} - C^{(l)}\|_F \leq 2\|\mathfrak{C}\|_F \cdot \mathcal{L}ip(h_{\theta^{(l)}})\epsilon^{(l)}\|X^{(l)}\|_F$$

Combining these two inequalities, we finally have,

$$\|\widehat{X}_B^{(l+1)} - X_B^{(l+1)}\|_F \leq \epsilon^{(l)} \cdot (1 + O(\mathcal{L}ip(h_{\theta^{(l)}})))\mathcal{L}ip(\sigma)\|C^{(l)}\|_F\|X^{(l)}\|_F\|W^{(l)}\|_F.$$

which concludes the proof. $\square$

Proof of Corollary 3 The proof is similar to the proof of Theorem 2.

Proof of Corollary 3: This time, we use the message passing equation for the back-propagation process, Eq. (3.3), restated as follows,

$$\nabla_{X^{(l)}}\ell = \sum_s \left(C^{(l,s)}\right)^\mathsf{T} \left(\nabla_{X^{(l+1)}}\ell \odot \sigma'\left(\sigma^{-1}\left(X^{(l+1)}\right)\right)\right) \left(W^{(l,s)}\right)^\mathsf{T}.$$

Note that,

$$\|\nabla_{X^{(l+1)}}\ell \odot \sigma'\left(\sigma^{-1}\left(X^{(l+1)}\right)\right)\|_F \leq \sigma'_{\max}\|\nabla_{X^{(l+1)}}\ell\|_F,$$

The rest of the proof simply follows the proof of Theorem 2, we similarly obtain,

$$\|\widehat{\nabla}_{X_B^{(l)}}\ell - \nabla_{X_B^{(l)}}\ell\|_F \leq \epsilon^{(l)} \cdot (1 + O(\mathcal{L}ip(h_{\theta^{(l)}}))\sigma'_{\max}\|C^{(l)}\|_F\|\nabla_{X^{(l+1)}}\ell\|_F\|W^{(l)}\|_F,$$

which concludes the proof. $\square$

3.8.3 More Theoretical Discussions

Upper-bound the estimation error of learnable parameters' gradients. Given that the gradients of each node in each layer is approximated with bounded error (Corollary 3), it is not hard to see that the gradients of learnable parameters $W^{(l)}$ and $\theta^{(l)}$ are also estimated with bounded errors. Firstly, the gradients of $W^{(l)}$, $\nabla_{W^{(l)}}\ell$, can be calculated from $\nabla_{X^{(l+1)}}\ell$ before going into the approximated back-propagation process. Secondly, the gradients of $\theta^{(l)}$, $\nabla_{\theta^{(l)}}\ell$ has bounded estimation error as long as for any $(i, j) \in \{1, \ldots, n\}^2$ that $|C'^{(l)}_{i,j} - C^{(l)}_{i,j}|$ is bounded, as we have shown in Section 3.8.2.

Upper-bound the size of VQ codebook. In order to effectively upper-bound the size of the VQ codebook, i.e., the number of reference vectors in VQ and the number of clusters in *k-means*, we generally need some extra mild assumptions on the distributions of the (hidden) features and gradients. For instance, if we assume the distributions of each feature and gradient in the l-th layer are *sub-Gaussian*, i.e., they have strong tail decay dominated by the tails of a Gaussian. Then, we can show the relative error of VQ $\epsilon = O(k^{-1/f})$, where f is the dimensionality of features and gradients. In addition to this, if we use product VQ (see Section 3.8.5) to utilize multiple VQ process, each working on a subset of $f_{\text{prod}} \ll f$ features and gradients, then we can bound the size of codebook as $k = O(\epsilon^{-f_{\text{prod}}})$.

Justification of choosing VQ as the dimensionality reduction method. We choose VQ as the method of dimensionality reduction to scale up GNNs because of the following reasons:

- Categorical constraint of VQ: We choose VQ mainly because of its categorical constraint, i.e., the rows of projection matrix R are unit vectors, shown in Eq. (5) of the manuscript. Intuitively, this means each node feature vector corresponds to exactly one codeword vector at a time, and thus we can replace its feature vector with the corresponding codeword. This is what we mean by ”node-identity-preserving” in Sections 3.3 and 4.3.

- Compared with PCA: PCA is not suitable when we want to compress the feature table of n nodes into a compact codebook when the number of nodes n is much larger than the number of features per node. Moreover, PCA is shown to have the same objective function as VQ under some settings but without the categorical constraint [84]. In PCA, each node feature is represented using the set of principal components (i.e., eigenvectors of the covariance matrix). Thus, we must use the complete set of principal components to recover each node feature vector before passing it to GNNs. Moreover, it is much harder to develop an online PCA algorithm to be used along with the training of GNNs.

- Compared with Fisher’s LDA: Fisher’s LDA, compared with PCA, is supervised and takes class labels into consideration. However, node classification or link prediction on a large graph is a semi-supervised learning problem, and we do not have access to all the node labels during training. We do not know how to compress the test nodes’ features under the transductive learning setting, and thus LDA is not a choice.

- Compared with randomized projection and locality-sensitive hashing: VQ is a better choice than randomized projection and locality-sensitive hashing because it is deterministic. Using VQ, we do not have to deal with the extra burden introduced by stochasticity.

3.8.4 More Discussions of Related Work

In this section, we continue the discussions of related work in Section 3.5 and review some other scalable or efficient methods for GNNs and beyond.

Other efficient methods for graph representation learning: MLP-based sample models and mixed–precision approaches. There exist some other MLP-based models, e.g., SGC [85], which requires only a one-time message passing during the pre-computation stage with $O(Lmf)$ time. However, SGC, PPRGo [86] and SIGN [87] over-simplify the GNN model and limit the expressive power. Despite the fact that they are fast, their performance is not generally comparable with other GNNs. Degree-Quant [88] and SGQuant [89] applies mixed-precision techniques specifically designed for GNNs to improve efficiency. Although Degree-Quant can reduce the runtime memory usage from 4x to 8x (SGQquant achieves 4.25-31.9x reduction), they did not provide a solution to effectively mini-batch the GNN training. We note that Degree-Quant and SGQuant do not provide means to mini-batch a large graph and are still "full-graph" training.

Outside graph learning: VQ in neural network and efficient self-attention. The general idea of using Vector Quantization (VQ) in a neural network is initially proposed for Variational Auto-Encoders (VAEs) in VQ-VAE [81, 90], and is later generalized to other generative modeling [91], contrastive learning [92], and self-supervised learning [93, 94]. Our work learns form their success and is one of the first attempts of applying VQ to large-scale semi-supervised learning. Our work also shares similarities with the recent advances of efficient self-attention techniques to speed up Transformer models. Linformer [95] linearizes the time and memory complexities of self-attention by projecting the inputs to a compact representation and approximating the self-attention scores by a low-rank matrix. Hamburger [96] generally discussed the update rule and back-propagation problem of VQ.

3.8.5 Algorithm Details

The complete pseudo-code. Here, we present the complete pseudo-code of our VQ-GNN algorithm as Algorithm 2. Three important components of VQ-GNN, namely *approximated forward message-passing*, *approximated backward message-passing*, and *VQ Update*, are highlighted in Algorithm 2 and in Fig. 3.3 in Section 4.3.

Product VQ and VQ-update rule. As described in Section 4.3, we basically follow the exponential moving average (EMA) update rule for codewords as proposed in [81]. In addition, we propose two further improvements:

1. Product VQ: we divide the $2f$-dimensional features concatenated with gradients into several f_{prod}-dimensional blocks. And apply VQ to each of the blocks independently in parallel.
2. Implicit whitening: we whitening transform the input features and gradients before using them for VQ update, exponentially smooth the mini-batch mean and variance of inputs, and inversely transform the learned codewords using the smoothed mean and variance.

In practice, the product VQ technique is generally required for VQ-GNN to achieve competitive performance across benchmarks and using different GNN backbones. Whitening and Lipschitz regularization are tricks that are helpful in some cases (for example, Lipschitz regularization is only helpful when training GATs). The three techniques are not introduced by us and are already used by some existing work related to VQ but outside of the graph learning community. For example, product VQ is used in [97], and whitening is used in [98].

The complete pseudo-code of VQ-update is Algorithm 3. It is important to note that in the experiments, we observed that implicit whitening helps stabilize VQ and makes it more robust across different GNN models and graph datasets. However, we observed that the smoothing of mini-batch mean and variance of gradients (which is used by the approximated backward message passing) is not compatible with some optimization algorithms which consider the cumulative history of gradients, e.g., *Adam*. This practical incompatibility is solved by using *RMSprop* instead of *Adam*

Dealing with the row-wise normalized learnable convolutions. As mentioned in Section 3.2 and Section 3.8.1, some self-attention based GNNs including GAT and Graph Transformers require further row-wise normalization of the convolution matrix as $C_{i,j}^{(l,s)} \leftarrow C_{i,j}^{(l,s)} / \sum_j C_{i,j}^{(l,s)}$. However, such a procedure is not characterized by the approximated message passing design in Section 4.3. Some special treatment is required to normalize the message weights passed to each target node. Actually, this normalization process can be realized by the following three steps:

1. Padding an extra dimension of ones to the messages, $X^{(l)} \leftarrow X^{(l)} \,||\, \mathbf{1}_n$
2. Message passing using the unnormalized convolution matrix
3. Normalization by dividing the last dimension, $X_{i,1:f}^{(l+1)} \leftarrow X_{i,1:f}^{(l+1)} / X_{i,f+1}^{(l+1)}$ for any $i = 1, \ldots, n$.

In this regard, we decouple the normalization of message weights with the actual message passing process. At the cost of an extra dimension of features (and thus gradients), we can VQ the message passing process again as we did in Section 4.3. In practice, we found this trick works well, and the experiment results on GAT and Graph Transformer in Sections 3.6 and 3.8.7 are obtained using this setup.

Regularizing the Lipschitz constants of learnable convolutions. Since our error bounds on the approximated features and gradients (Theorem 2 and Corollary 3) rely on the Lipschitz constant of learnable convolutions, and the "decoupled row-wise normalization" trick discussed above requires some means to control the unnormalized message weights, we have to Lipschitz regularize some learnable convolution GNNs including GAT and Graph Transformers. We follow the approach described in [99] to control the Lipschitz constant of GAT and Graph Transformer without affecting their expressive power. Please see [99] for details.

Non-linearities, dropout, normalization. In experiments, we found our algorithm, VQ-GNN, is compatible with any non-linearities, dropout, and additional batch- or layer-normalization layers.

3.8.6 Implementation Details

Hardware specs. Experiments are conducted on hardware with Nvidia GeForce RTX 2080 Ti (11GB GPU), Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, and 32GB of RAM.

Dataset statistics. Detailed statistics of all datasets used in the experiments are summarized in Table 3.6.

Table 3.6: Information and statistics of the benchmark datasets.

Dataset	*ogbn-arxiv*	*Reddit*	*PPI*	*ogbl-collab*	*Flickr*
Task	node	node	node	link	node
Setting	transductive	transductive	inductive	transductive	transductive
Label	single	single	multiple	single	single
Metric	accuracy	accuracy	F1-score	hits@50	accuracy
# of Nodes	169,343	232,965	56,944	235,868	89,250
# of Edges	1,166,243	11,606,919	793,632	1,285,465	449,878
# of Features	128	602	50	128	500
# of Classes	40	41	121	(2)	7
Label Rate	54.00%	65.86%	78.86%	92.00%	50.00%

We note from Table 3.6 that:

- *PPI* is a node classification benchmark under the inductive learning setting, i.e., neither attributes nor connections of test nodes are present during training.
- *PPI* benchmark comes with multiple labels per node, and the evaluation metric is F1 score instead of accuracy.
- *Flickr* and *Reddit* have 500 and 602 features per node, respectively. The increased dimensionality of input node features may be challenging for VQ-GNN because VQ has to compress higher-dimensional vectors. Moreover, *Reddit*'s average node degree is 49.8. More memory is required for mini-batches of the same size because more messages are passed in a layer of GNN.

Hyper-parameter setups of VQ-GNN. To simplify the settings, we fix the hidden dimension of GNNs to 128 and layer number to 3 across the experiments. We set the size of the VQ codewords to 1024, and its size as small as 256 should also work well. We choose RMSprop (alpha=0.99) as the optimizer, and the learning rate is fixed at 3e-3. Batch norm is used for stable training. We do not use drop out for either our method or the baselines.

Hyper-parameter setups of other scalable methods. For baseline models, we follow the practice of OGB. The optimizer is Adam, and the learning rate is fixed at 1e-3. For a fair comparison with respect to memory consumption, on the `ogbn-arxiv` dataset we set hyper-parameters as below: SAGE-NS with the batch size 85K, per-layer sampling sizes [20,10,5]; ClusterGCN batch size 80, number of partitions 40; GraphSAINT-RW batch size 40K, walk-length 3, number of steps 2. We use the hyper-parameter setting for experiments in Table 3.3 (the fixed node setting) and Figure 3.4. The parameter setting ensures to traverse over all the nodes in the graph in one epoch of training. For fixed message setting in 3.3, we alter the batch size of SAGE-NS to 35K, ClusterGCN to 60, GraphSAINT-RW to 60K. Here the batch size for ClusterGCN is small because each batch item means a cluster group of nodes.

Multi-branch. To mitigate the error induced by VQ in the high-dimensional space, we split feature vectors into small pieces. In practice, we find that when the dimension of each piece is 4, our algorithm generally works well. When the split dimension is 4 we have 32 separate branches each layer to do the VQ. In this manner each layer we will do 32 times of VQ and convolution independently. Having more branches will slow down the process of training but will provide more accurate quantized features. At the end of each layer, separated feature vectors are concatenated together to restore the original hidden dimension, and the restored feature is input to the next layer. We view this separate convolution implementation as V1. The other implementation of VQ-GNN is to follow a recent parallel work, GNNAutoScale [100], to reconstruct the node features of the out-of-mini-batch neighbors for a mini-batch using the learned codewords, and then perform forward-pass message passing between the with-in-mini-batch nodes and the reconstructed nodes. This implementation is more straightforward but may suffer from larger memory overhead when the underlying graph is dense, e.g., on the *Reddit* benchmark. We call this second implementation V2. V1 and V2 only differ in terms of memory consumption, training speed, and the sampling methods. By default V2 uses cluster sampler as GNNAutoScale [100] to restrict the growth of the size of out-of-batch nodes. In the paper, results on *ogbn-arxiv*, *PPI*, and *ogbl-collab* are given by V2. Results on *Reddit* and *Flickr* are by V1. We open source both implementations in our code release.

3.8.7 Ablation Studies and More Experiments

Experiments on the *Flickr* benchmark We conduct another set of performance comparison experiments on the *Flickr* node classification benchmark in Table 3.7, whose information and statistics are listed in Table 3.6. As in Section 3.6, we can draw a similar conclusion that VQ-GNN shows more robust performance than the three scalable baselines.

Table 3.7: Performance comparison between sampling-based baselines and our approach, VQ-GNN, on the *Flickr* benchmark.

Task Benchmark	Node Classification (Transductive) *Flickr* (Acc.±std.)		
GNN Model	GCN	SAGE-Mean	GAT
"Full-Graph"	$.5209 \pm .0053$	$.5177 \pm .0041$	$.5156 \pm .0067$
NS-SAGE.	NA	$.5165 \pm .0077$	$.5282 \pm .0052$
Cluster-GCN	$.4976 \pm .0078$	$.4996 \pm .0045$	$.4907 \pm .0107$
GraphSAINT-RW	$.5239 \pm .0071$	$.5040 \pm .0046$	$.5163 \pm .0062$
VQ-GNN (Ours)	$.5315 \pm .0031$	$.5323 \pm .0083$	$.5288 \pm .0054$

Ablation studies We also conduct several ablation experiments on the *ogbn-arxiv* benchmark with GCN backbone. Please note that if not mentioned otherwise, the hyper-parameter setups will follow the corresponding model and are described in Section 3.8.6.

- **Performance vs. the number of layers**: We vary the number of layers of GCN backbone from one to five, and the performance of VQ-GNN on *ogbn-arxiv* is shown in the following table. We see that a two-layer GCN is sufficient to achieve good performance on *ogbn-arxiv* while using only one layer harms the performance. Stacking more layers will not bring any performance gain.

# of Layers	1	2	3	4	5
Accuracy	0.6599	0.7006	0.7055	0.7080	0.7037

- **Performance vs. codebook size**: We vary the codebook size from 64 to 4096 and fix the mini-batch size at 40K. Performance is shown as follows, where we can see the performance is not sensitive to the codebook size. Setting the codebook size to 64 is enough to achieve relatively good performance on *ogbn-arxiv*. This may indicate that the node feature distribution of *ogbn-arxiv* is sparse, which is reasonable as the 128-dimensional node features are obtained by averaging the embedding of words in the titles and abstracts of arXiv papers [12].

Codebook Size	64	256	1024	4096
Accuracy	0.6950	0.7011	0.7030	0.7049

- **Performance vs. mini-batch size**: We vary the mini-batch size from 10K to 80K, and the performance is shown as follows. We see that smaller mini-batch sizes can slightly lower the overall performance. When the mini-batch size is smaller, more messages come from out-of-mini-batch nodes (see Fig. 3.1), and they are approximated by the VQ codewords. This increased number of approximated messages can harm the performance. However, generally speaking, the performance is not sensitive to the mini-batch size as long as it is not small.

Mini-batch Size	10K	20K	40K	60K	80K
Accuracy	0.6843	0.6920	0.7011	0.7055	0.7061

- **Performance vs. mini-batch sampling strategy**: We compare three different sampling strategies: (1) randomly sampling nodes, (2) randomly sampling edges, and (3) sampling using random walks as in GraphSAINT-RW. As shown in the following table, we did not observe an obvious difference in the performance.

Mini-batch Sampling Techniques	Sampling Nodes	Sampling Edges	Sampling using Random Walks
Accuracy	0.7020	0.7034	0.7023

Performance of VQ-GNN with Graph Transformers [68]. Our algorithm enables graph transformer architectures to compute global attention on large graphs. For each layer, we input hidden features into VQ-GNN, Graph Transformer, and Linear modules separately and sum the output features of each module together. In this way, the holistic model can not only leverage global attention but can also absorb local information. The transformer module is adapted from [68]. We show the performance in Table 3.8. We find that the removal of the batch norm will mitigate the problem of overfitting, so our model does not involve batch norm.

Table 3.8: Performance of VQ-GNN with Graph Transformer Backbone on the *ogbn-arxiv* benchmark.

Benchmark	*ogbn-arxiv* (Acc.±std.)
Global Attention + GAT [68]	$.7108 \pm .0055$

Algorithm 2: VQ-GNN: our proposed universal framework to scale most state-of-the-art Graph Neural Networks to large graphs using Vector Quantization. For ease of presentation, we assume the GNN has only one fixed convolution matrix.

Require: Input node features X, ground-truth labels Y
Require: GNN's convolution matrix C (fixed convolution) or $\mathfrak{C}$ and $h(\cdot,\cdot)$ (learnable convolution)

procedure VQ-GNN$_C(X, Y)$
  for $l = 0, \ldots, L-1$ **do** ▷ *Initialization*
    Randomly initialize GNN learnable parameters $W^{(l)}$ and $\theta^{(l)}$ and codewords $\widetilde{V}^{(l)} = \widetilde{X}^{(l)} \,||\, \widetilde{G}^{(l+1)}$ which are feature codewords $\widetilde{X}^{(l)}$ concatenated with gradient codewords $\widetilde{G}^{(l+1)}$
    Initialize codeword assignment $R^{(l)}$ according to the initial codewords
  end for
  for epoch $t = 1, \ldots, T$ **do**
    for indices $\langle i_b \rangle$ sampled from $\{1, \ldots, n\}$ **do** ▷ *Mini-batch Training*
      Load the mini-batch features $X_B (= X_{\langle i_b \rangle, :})$, labels Y_B, selected rows and columns of convolution matrix C_B and $(C^{\mathsf{T}})_B$, and selected rows of each codeword assignment matrix $R_B^{(l)}$ to the training device, and set $X_B^{(0)} \leftarrow X_B$
      for $l = 0, \ldots, L-1$ **do** ▷ *Forward-Pass*
        Compute the approximate message passing weight matrix $\mathscr{C}^{(l)}$ using $C_B, C_B^{\mathsf{T}}, R_B^{(l)}$; see Eq. (3.6)
        APPROXIMATED FORWARD MESSAGE-PASSING: estimate next layer's features $X_B^{(l+1)}$ using previous layer's mini-batch features $X_B^{(l)}$ and feature codewords $\widetilde{X}^{(l)}$; see Eq. (3.6)
      end for
      Compute $\ell = \text{LOSS}(Y_B, \hat{Y}_B)$, where the predicted labels $\hat{Y}_B = \text{SOFTMAX}(X_B^{(L)})$ (node classification) or $\text{LINKPRED}(X_B^{(L)})$ (link prediction), and set $G_B^{(L)} \leftarrow \nabla_{X_B^{(L)}} \ell$ (since no non-linearity σ in the last GNN layer; see Eq. (3.7))
      for $l = L-1, \ldots, 0$ **do** ▷ *Back-Propagation*
        APPROXIMATED BACKWARD MESSAGE-PASSING: estimate lower layer's gradients $G_B^{(l)}$ and $\nabla_{W^{(l)}} \ell$ using higher layer's mini-batch gradients $G_B^{(l+1)}$ and gradient codewords $\widetilde{G}^{(l+1)}$; see Eq. (3.7)
      end for
      for $l = 0, \ldots, L-1$ **do** ▷ *VQ Update*
        VQ UPDATE: Update the concatenated codewords $\widetilde{V}^{(l)} = \widetilde{X}^{(l)} \,||\, \widetilde{G}^{(l+1)}$ and codeword assignment $R_B^{(l)}$ using this mini-batch's concatenated feature and gradient vectors $V^{(l)} = X_B^{(l)} \,||\, G_B^{(l+1)}$ and the old concatenated codewords
        Synchronize the codeword assignment matrix $R^{(l)}$ stored outside the training device with the updated $R_B^{(l)}$
      end for
      Update learnable parameters $W^{(l)}$ using the estimated gradients $\nabla_{W^{(l)}} \ell$
    end for
  end for
end procedure

Algorithm 3: VQ-Update: our proposed algorithm to update VQ codewords and assignment using exponential moving average (EMA) estimates of codewords with implicit whitening of inputs.

Require: Input mini-batch vectors $V \in \mathbb{R}^{b \times f_{\text{prod}}}$ as a part of the $2f$-dim feature and gradients
Require: Exponential decay rate γ for momentum estimates of codewords
Require: Exponential decay rate β for momentum estimates of mini-batch mean and variance
Require: Codewords $\tilde{\bar{V}} \in \mathbb{R}^{k \times f_{\text{prod}}}$ before update
Require: Smoothed mean $\widetilde{\mathrm{E}}[V]$ and variance $\widetilde{\mathrm{Var}}[V]$ before update

function VQ-UPDATE$_{(\gamma,\beta)}(V, \tilde{\bar{V}})$ ▷ Update VQ for a f_{prod}-dim block
    Whitening transform V to $\bar{V}$ and get the mini-batch mean $\mathrm{E}[V]$ and variance $\mathrm{Var}[V]$
    $\widetilde{\mathrm{E}}[V] \leftarrow \widetilde{\mathrm{E}}[V] \cdot \beta + \mathrm{E}[V] \cdot (1-\beta)$ (EMA update of smoothed mean)
    $\widetilde{\mathrm{Var}}[V] \leftarrow \widetilde{\mathrm{Var}}[V] \cdot \beta + \mathrm{Var}[V] \cdot (1-\beta)$ (EMA update of smoothed variance)
    $R_B \leftarrow \textsc{FindNearest}(\bar{V}, \tilde{\bar{V}})$ (update assignment, $(R_B)_{i,v} = 1$ if $\tilde{\bar{V}}_{v,:}$ is closest to $\bar{V}_{i,:}$)
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} \cdot \gamma + R_B^{\mathsf{T}} \mathbf{1}_b \cdot (1-\gamma)$ (momentum update of cluster sizes)
    $\mathbb{C} \leftarrow \mathbb{C} \cdot \gamma + R_B^{\mathsf{T}} \bar{V} \cdot (1-\gamma)$ (momentum update of cluster vector sums)
    $\tilde{\bar{V}}_{v,:} \leftarrow \frac{1}{\boldsymbol{\eta}_v} \mathbb{C}_{v,:}$ (update codewords)
    Inversely whitening transform $\tilde{\bar{V}}$ to $\widetilde{V}$ using $\widetilde{\mathrm{E}}[V]$ and $\widetilde{\mathrm{Var}}[V]$
    return $\widetilde{V}$, R_B
end function

Chapter 4: GOAT: A Global Transformer on Large-scale Graphs

4.1 Introduction

Transformers [101] have demonstrated efficacy in many domains including language understanding and computer vision [102, 103], and this has spurred interest in applying transformers to the graph domain. However, the success of transformers for graphs has been more modest, and has mostly been in the fairly narrow regime of graph classification tasks like molecule classification [69, 104, 105, 106, 107, 108, 109]. The success of transformers in this regime is largely a result of the small size of each problem instance. For instance, the mean node count of graphs from the `ogbg-molhiv` dataset of the *Open Graph Benchmark* [12] is only 25.5. This tiny size enables self-attention across a larger percent of the graph, enabling *long-range* or even *global* self-attention. Despite a recent surge in interest in attention-based networks, standard Graph Neural Networks (GNNs) [1, 28] are still the de-facto model of choice for broader applications involving node classification or large industrial graphs.

Research in other domains suggests that a transformer's ability to utilize long-range signals that were previously inaccessible in more constrained sequential models are its key success factor. However, it is well-known that this self-attention is expensive, with time and memory overhead growing quadratically with the length of the input. To address this issue for language transformers, a range of efficient variants have been proposed [95, 110, 111, 112, 113] and have demonstrated competitive performance.

When applying transformers to graphs, sequence length is akin to the size l of the l-hop neighborhood, but the size grows exponentially instead of linearly. For large enough l, the model becomes *global* and attends to the entire graph. Tasks such as node classification are usually done on large-scale graphs such as `ogbn-products` (2.4M nodes). Attending to the entirety of this graph in a naive way would require 24TB of GPU memory [114]. It is an open and pressing challenge to design efficient graph transformer models that scale to graphs of this size. Existing works on graph transformers for node classification never go beyond recursive 1-*hop-neighbor* message passing [115, 116, 117] so they fail to learn from larger context and perhaps fail to achieve the full potential demonstrated in other domains.

At the same time, an established limitation of GNNs is their over-reliance on the homophily principle [118] causing them to perform poorly on heterophilious graphs. Homophily (or heterophily) means that a node's neighbors are likely (unlikely) to be of the same class. Standard GNNs are built on the message passing scheme [18], where features are aggregated recursively between 1-hop neighbors. This scheme has strong inductive bias towards homophilious graphs and does not tolerate heterophily well. To address this, researchers have proposed various heterophily-centered GNN models [2, 119, 120, 121]. These models usually involve specialized message passing schemes that do not work for homophilious graphs. GPR-GNN [122] can adapt to different homophily/heterophily profiles, but this model uses the entire graph for each training update ("full-batch" training) and thus cannot scale to huge problems. While it is reasonable to have specialized model designs for different kinds of graphs, this practice can be problematic when the homophily or heterophily characteristics of the target dataset are unknown, or the graph has mixed behavior.

Present work. We study a global transformer in which each node attends to all others, providing a universal architecture that supports both homophilious and heterophilious graphs. We propose GOAT, a global transformer for large-scale node classification tasks. To implement the intractable $O(n^2)$ global self-attention on large-scale graphs, we leverage a dimensionality reduction algorithm and reduce memory complexity from quadratic to linear. Using a K-Means based projection algorithm, we theoretically show that our scalable global attention method has bounded error relative to graph attention without dimensionality reduction. Besides the global design, we also strengthen the model using a scalable local attention module. For each node, we sample its l-hop neighbors and make it *directly* attend to them, unlike the recursive smoothing pattern of GNNs. Empirically, GOAT shows strong performance on both homophilious and heterophilious datasets as large as `ogbn-products` (2.4M nodes) [12] and `snap-patents` (2.9M nodes) [2]. We summarize our contributions as follows:

1. We propose GOAT, a scalable global transformer model where each node is able to attend to all others.

2. We develop a novel local attention module that enables GOAT to absorb rich local information.

3. We demonstrate the strong performance of GOAT on both large-scale homophilous and heterophilious node classification benchmarks.

4. We provide theoretical justification to show our scalable global attention scheme has bounded error relative to unscalable standard attention.

4.2 Preliminaries

In this section we introduce the preliminaries of GNNs, transformers, and homophily.

Graph Neural Networks (GNNs). We represent a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E})$. GNNs are often built with recursive message-passing schemes, where features are passed and shared directly between 1-hop neighbors. Formally the k-th iteration of message passing, or the forward propogation of the k-th layer of GNNs, is defined as follows:

$$\begin{aligned} m_v^{(k)} &= \text{AGGREGATE}^{(k)}\left(\left\{\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}\right), \forall u \in \mathcal{N}(v)\right\}\right), \\ h_v^{(k)} &= \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, m_v^{(k)}\right), \end{aligned} \tag{4.1}$$

where $h_v^{(k)}$ is the hidden feature of node v at the k-th layer, $m_v^{(k)}$ is the computed message for node v at the k-th layer, e_{uv} is the edge feature between node u and v, $\mathcal{N}(v)$ is node v's 1-hop neighbor set. AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ are functions parameterized by neural networks.

Transformers. The key component of a transformer model is the self-attention scheme, which allows each element of a set or token to attend to information of other tokens at various locations. The forward pass of a self-attention module is defined as:

$$\mathrm{Attn}(H) = \mathrm{Softmax}\left(\frac{HW_Q\left(HW_K\right)^\top}{\sqrt{d}}\right)HW_V, \tag{4.2}$$

where $H \in \mathbb{R}^{n\times f}$ is the hidden feature matrix. $W_Q, W_K, W_V \in \mathbb{R}^{f\times d}$ are linear projection matrices with trainable weights. Here the Equation 4.2 denotes the single-head self-attention module, which can straightforwardly generalize to multi-head attention. Note that in practice multi-headed self-attention is widely used. It is easy to see that attention is expensive. The time and memory complexity is $O(n^2)$, which leads to low efficiency and is a bottleneck for transformers. In this work, we call the output of the Softmax function the "attention matrix".

Homophily indicates connected nodes are likely to share common labels. We follow Lim et al. [2] to focus on edge homophily in this work. The edge homophily is defined as the proportion of edges that link two nodes with the same label as below:

$$h = \frac{|\{(u,v) \in \mathcal{E} : y_u = y_v\}|}{|\mathcal{E}|}, \tag{4.3}$$

where $\mathcal{E}$ is the edge set and y is the node label. Non-homophily (or **heterophily**) graph indicates the dataset with dissimilar labels/features sharing edges.

4.3 Method

Intuition. Existing GNNs usually have hardcoded message-passing patterns and will only work on either homophilious or heterophilious graphs. A global attention scheme makes each node attend to all the nodes in the graph and does not explicitly have inductive bias towards either one. Instead of one specially tailored or fixed message passing and aggregating pattern, the attention scheme freely learns to adapt to different priors. Furthermore, the ability to attend to a single far-away node may enable multi-hop features to be used without the over-smoothing that happens when information is passed over many rounds of averaging in a standard GNN. However, for applications in industry where large graphs with millions of nodes are ubiquitous, it is not possible to train and deploy a fully global transformer due to the quadratic cost.

Given that, we propose **GOAT**, the scalable global transformer. GOAT uses an approximate global attention which reduces complexity from quadratic to linear and supports mini-batch training. GOAT also has a local attention module to process information from the local neighborhood for better prediction. Figure 4.1 illustrates the local sampling procedure and the whole attention module. In this section, we describe our methodological designs in detail. Firstly let us have a formal definition on the notations.

Definition 1. *We define n as the total number of nodes in the graph, k is the number of dimension in the low dimensional space, $k \ll n$. P denotes the projection matrix, $P \in \mathbb{R}^{n\times k}$. f denotes the raw feature dimension of the node feature and d is the hidden feature dimension if without further specification. b denotes batch size.*

Global. To address the expensive quadratic complexity issue, we leverage *dimensionality reduction*. Firstly we want to find a projection matrix P that can reduce the feature matrices into a low-dimensional space so the cost of computing their product is reduced. At the same time we also hope this reduction will not overly degrade the quality of representations. Below, we provide theoretical analysis on the existence of such matrix, P. Note that in the demonstration process below, we assume positional information is already fused into node features.

Theorem 4. *For any linear weight matrices $W_K, W_Q, W_V \in \mathbb{R}^{f\times d}$, node feature matrix $X \in \mathbb{R}^{n\times f}$, mini-batch of nodes $X_B \in \mathbb{R}^{b\times f}$, row vector $x \in \mathbb{R}^{1\times n}$ of X, and $\varepsilon > 0$, there exists projection matrices $P_A, P_V \in \mathbb{R}^{n\times k}$, such that*

$$\begin{aligned}\Pr(&\left\|\operatorname{Softmax}\left(SP_A\right)P_V^\top XW_V - \operatorname{Softmax}(S)XW_V\right\|_F \\ &\leqslant \varepsilon \|\operatorname{Softmax}(S)\|_F \left\|XW_V\right\|_F) > 1 - O(1/n),\end{aligned} \tag{4.4}$$

with $S = X_B W_Q \left(XW_K\right)^\top / \sqrt{d}$ and $k = O\left(\log(n)/\varepsilon^2\right)$.

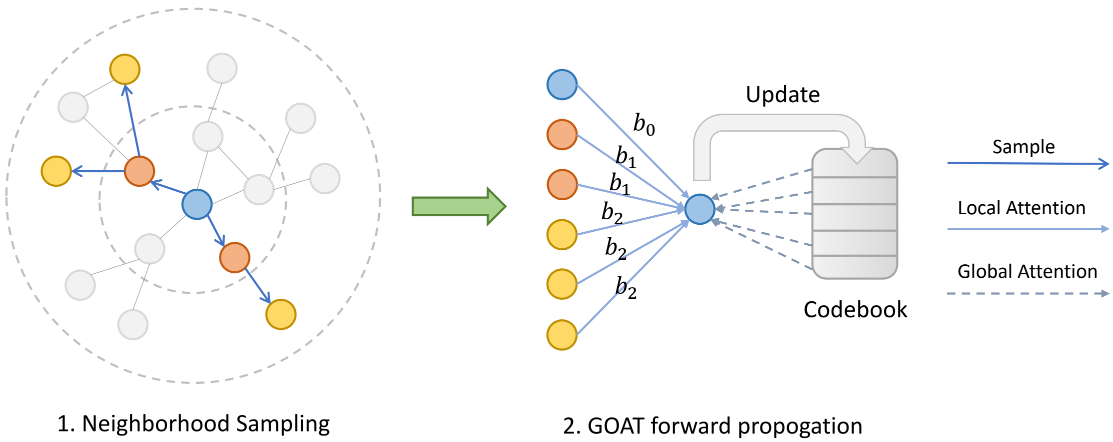


Figure 4.1: The local sampling procedure and forward propogation of the GOAT model. b_l denotes the trainable positional bias for neighbors at a distance of l.

We argue that the scheme provided by Theorem 4 makes global attention possible on large graphs. The expression $SP_A = \left(X_B W_Q (XW_K)^\top / \sqrt{d}\right) P_A$ inside the Softmax function can be computed as a product between $Q_B = X_B W_Q$ and $\widetilde{K} = P_A^\top X W_K$, which is only $O(bdk)$ work. Note that the new value feature matrix $\widetilde{V} = P_V^\top X W_V$ is also low dimensional. Then we see that the problem comes to how we can materialize $\widetilde{K}, \widetilde{V}$ without explicitly computing the multiplication of $(XW_K)^\top P_A$ or $P_V^\top X W_V$ which is $O(nfd + nkd)$ and not scalable when the graph is large.

Technically, we follow Van Den Oord et al. [123] to materialize $\widetilde{K}$ and $\widetilde{V}$. Projection matrix P is computed by the K-Means algorithm as the sparse indexing matrix. Each row of $P \in \mathbb{R}^{n \times k}$ is a one-hot vector representing the clustering centroid the node is assigned to. We define

$$C = \text{diag}^{-1}(\mathbf{1}_n P) P^\top X$$

as the *codebook*. Each row of the codebook C represents the center (e.g., the average) of all the entries in a cluster. In this way, $\widetilde{K} = \text{diag}^{-1}(\mathbf{1}_n P) P^\top X W_K = C W_K$ and $\widetilde{V} =$

$\mathrm{diag}^{-1}(\mathbf{1}_n P)P^\top X W_V = C W_V$. We store and update $\widetilde{K}, \widetilde{V}$ in the exponential moving average (EMA) manner. The algorithm is summmarized in Algorithm 5. Note that we do not explicitly compute $\widetilde{K}, \widetilde{V}$ each iteration but instead cache and update the codebook on the fly in the EMA manner using batch statistics. P is stored sparsely to save memory.

Now we go back to show that the forward pass under such a scheme yields a bounded-error approximation compared with the authentic output. We start with two definitions for clearer demonstration. We define

$$\widetilde{X} = P\mathrm{diag}^{-1}(\mathbf{1}_n P)P^\top X = PC, \tag{4.5}$$

as the approximate X using the codebook. We define the attention matrix for a batch X_B computed by parameterized function f_W as

$$f_W(Y) = \mathrm{Softmax}\left(\frac{X_B W_Q \left(Y W_K\right)^\top}{\sqrt{d}}\right). \tag{4.6}$$

Theorem 5. *If the function f_W has Lipschitz constant upper-bounded by $\mathrm{lip}\left(f_W\right)$ and the quantization error is ε, the estimation error is bounded as,*

$$\left\|\widetilde{X}_B^{out} - X_B^{out}\right\|_F \leq \varepsilon \cdot \left[1 + O\left(\mathrm{lip}\left(f_W\right)\right)\right] \|A_B\|_F \cdot \|X\|_F \cdot \|W_V\|_F, \tag{4.7}$$

with $\widetilde{X}_B^{out} = \widetilde{A}_B \widetilde{X} W_V$, $X_B^{out} = A_B X W_V$, $A_B = f_W(X) = \mathrm{Softmax}\left(X_B W_Q \left(X W_K\right)^\top / \sqrt{d}\right)$, and $\widetilde{A}_B = f_W(\widetilde{X}) = \mathrm{Softmax}\left(X_B W_Q (\widetilde{X} W_K)^\top / \sqrt{d}\right)$. Quantization error ε is defined as $\|X - \widetilde{X}\|_F \leq \varepsilon \|X\|_F$.

Note that the computation of $\widetilde{X}_B^{\mathrm{out}}$ denoted in Theorem 5, which is the approximated global attention, is expensive because it requires the computation of matrix multiplication of $X_B W_Q \widetilde{X} W_K$ and also $\widetilde{A}_B \widetilde{X} W_V$, which is unscalable. We articulate it in this way for the simplicity of theoretical

Algorithm 4: Global Transformer mini-batch forward propogation algorithm

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; $\mathrm{MLP_a}$ and $\mathrm{MLP_b}$ are two independent MLPs; PE is the precomputed positional encoding based on the structure of graph $\mathcal{G}$; μ_x, μ_{pe} are the centroids computed by the K-Means algorithm; P is the centroid assignment index for each node; W_Q, W_K, W_V are trainable parameters.

for $v \in \mathcal{V}$ **do**

  $x \leftarrow \mathrm{MLP_a}(X_v)$

  $pe \leftarrow PE_v$

  $q \leftarrow \mathrm{Concat}(x, pe)W_Q$

  $K \leftarrow \mathrm{Concat}(\mu_x, \mu_{pe})W_K$

  $V \leftarrow \mu_x W_V$

  $x_{out} \leftarrow \mathrm{Softmax}\left(\frac{qK^\top}{\sqrt{d}} + \log\left(\mathbf{1}_n P\right)\right) V$

  $x_{out} \leftarrow \mathrm{MLP_b}(x_{out})$

  Update μ_x, μ_{pe} by x, pe using the EMA K-Means algorithm as in Algorithm 5.

end for

proof. Below we show that $\widetilde{X}_B^{\mathrm{out}}$ can be equivalently computed using $\widetilde{K}$ and $\widetilde{V}$, which are low-dimensional.

Corollary 6. *The computation of* $\widetilde{X}_B^{out} = \mathrm{Softmax}\left(X_B W_Q (\widetilde{X} W_K)^\top / \sqrt{d}\right) \widetilde{X} W_V$ *is equivalent with*

$$\widetilde{X}_B^{out} = \mathrm{Softmax}\left(X_B W_Q \widetilde{K}^\top / \sqrt{d} + \log\left(\mathbf{1}_n P\right)\right) \widetilde{V}. \tag{4.8}$$

All the proofs are deferred to the Appendix.

Through Corollary 6, where X_B gets to directly attends to $\widetilde{K} = CW_K$ instead of $\widetilde{X}$. As we have discussed, both $\widetilde{K}$ and $\widetilde{V}$ will be materialized in memory and updated in the EMA way. The computation for the feature of a node $\widetilde{X}_B^{\mathrm{out}}$ is $O(bdk)$ and thus scalable and efficient. Algorithm 4 illustrates our global transformer forward pass.

It is well established in the literature that good positional encodings are required to make transformers work effectively. In our global attention scheme, only absolute positional encodings are feasible because the hidden features and positional encodings must be concatenated. We

Algorithm 5: EMA K-Means update algorithm

Require: Inputs are the hidden features X and positional encodings PE for a batch. $\mathrm{bn}(\cdot)$ is the batch norm module. $\mathrm{FindNearest}(\cdot)$ finds the nearest centroid for each feature.

def Update(X, PE):

$F \leftarrow \mathrm{Concat}(X, PE)$

$F \leftarrow \mathrm{bn}(F)$ ▷ data whitening using batch norm

$\mu \leftarrow \mathrm{Concat}(\mu_x, \mu_{pe})$

$P \leftarrow \mathrm{FindNearest}(F, \mu)$ ▷ compute cluster assignment

$c \leftarrow c \cdot \gamma + P^\top \mathbf{1} \cdot (1 - \gamma)$ ▷ EMA accumulation

$v \leftarrow v \cdot \gamma + P^\top F \cdot (1 - \gamma)$ ▷ EMA accumulation

$v \leftarrow v/c$

$\mu \leftarrow v \cdot \mathrm{bn.\,running\_std} + \mathrm{bn.\,running\_mean}$

$\mu_x, \mu_{pe} \leftarrow \mu$

argue that the design of positional encoding on graphs is still an open question for the community [108, 124] and detailed discussion for such design is beyond the scope of this work. To simplify the setup of the experiments, we use pretrained node2vec [125] node embeddings as our positional embeddings.

Local. Along with the design of our global transformer, we propose a novel local attention module, which allows each node to *directly* attend to its l-hop neighbors to attain rich local information. Empirically, we find that the local attention module effectively helps the model learn better representations. Although this module is responsible for aggregating information from the *local* neighborhood as in normal message passing feature aggregation, unlike the standard GNNs' recursive 1-hop neighbor smoothing pattern, our local module provides flexible attention weights for neighbors at different hop distances. We believe the module provides increased capacity to learn the inductive biases required for accurate predictions beyond the hard-coded structures of existing GNNs. To support mini-batch training, we adopt the widely-used neighbor sampling (NS) method [17] to sample all l-hop neighbors. We make each node directly attend to the sampled neighbors. Inspired by Ying et al. [104], for the local module we select the *relative* positional encoding scheme to distinguish neighbors at diverse distances from the source node. Our attention score calculation inside the Softmax function is:

$$S_{ij} = \frac{X_i W_Q \left(X_j W_K\right)^{\top}}{\sqrt{d}} + b_{D(i,j)}, \tag{4.9}$$

where $b_{D(i,j)}$ is a trainable bias parameter indexed by $D(i, j)$, the shortest distance between node i and j. Note that although the local module intuitively helps learn better representations, the neighbor sampling (NS) algorithm does introduce a small scalability concern. It is well-known that the neighborhood explosion problem for NS has complexity $O(d^l)$ for l-hop neighborhoods, where d is the average degree of the graph. Surprisingly, it is the NS step, and not the approximate global attention, that is the primary efficiency bottleneck for our model. We investigate the efficiency perspective in Section 4.5.

Since we leverage different positional encodings for the global and local modules, we have separate attention functions, after which the respective sets of node features and positional encodings are concatenated and fed into subsequent layers. Figure 4.1 illustrates the local sampling procedure and the whole attention module. In the sections to follow, we show that GOAT displays strong performance on the large-scale node classification task.

4.4 Experiments

In this section, we detail the empirical evaluation of our GOAT model.

Datasets. We select four datasets to evaluate. To represent homophilious graph problems we choose two datasets, `ogbn-arxiv` and `ogbn-products`, from the well-known *Open Graph Benchmark* (OGB) [12]. For heterophilious examples we utilize the `arxiv-year` and

Table 4.1: Dataset statistics.

Dataset	# Nodes	# Edges	# Feat.	# Class	Class type	Split	Edge hom.
ogbn-arxiv	169,343	1,166,243	128	40	product category	official	.66
ogbn-products	2,449,029	61,859,140	100	47	subject area	official	.81
arxiv-year	169,343	1,166,243	128	5	pub year	random	.222
snap-patents	2,923,922	13,975,788	269	5	time granted	random	.073

`snap-patents` datasets curated by Lim et al. [2]. These are all large-scale (multi-million) node classification datasets. For the train and validation splits, we use the official splits from OGB for both `ogbn-arxiv` and `ogbn-products` and for `arxiv-year` and `snap-patents` we follow the practice of Lim et al. [2] and randomly sample the train and validation sets. As there are no official splits or train set ratios for these two datasets, we experiment with training sets that comprise 10%, 20%, and 50% of the data while fixing validation set ratio at 25%, and report separate results for each split. We refer readers to Table 4.1 for detailed statistics of the datasets.

Setup. We focus on the transductive node classification task, where we see all the nodes at training time, but only the train set has labels. Our baseline models include GCNJK [126], GAT [28], LINKX [2], MixHop [119], and GPS [127]. All of our training procedures use pure empirical risk minimization (ERM) and we do not leverage techniques like data augmentation [128], label propogation [129], powerful embeddings [130], or other tricks, as these additional regularizers have not been uniformly studied for all model types, and this simple setting enables fair comparisons across model architectures. For the baseline models, we perform a hyperparameter sweep and select the best performing settings and report the corresponding results for each model and dataset pairing (full details in the Appendix). For GOAT, the attention function is multi-headed but we only implement a single layer of attention module. We leave the discussion of a multi-layer variant to future work. For the local attention, we sample neighbors that live within 3-hops. For each node we sample $[20, 10, 5]$ neighbors recursively. The size of the codebook is fixed at $4,096$ and the dimensionality is 64. We always use a dropout rate of 0.5 and also use batch norm. Each experiment is carried out on either a single GeForce RTX 2080 Ti (11GB memory) or a RTX A4000 (16GB memory).

Results. Table 4.2 reveals the competitive performance of our GOAT model compared with baselines. For * we show performance of MixHop with GraphSAINT sampling [31] on `snap-patents` despite Lim et al. [2] reporting 46.82 ± 0.11 for MixHop with ClusterGCN sampling [30] as their runner-up to LINKX on this dataset. This setting was OOM/T on our hardware, but as it is weaker than LINKX and GOAT in either case, reporting the stronger numbers would not have changed the results or analysis of average model performance. GCNJK and GAT intuitively register strong performances on `ogbn-arxiv` and `ogbn-products` as they resonate with the inductive bias of the two homophilious datasets. In contrast, on the `arxiv-year` and `snap-patents` datasets, their scores are poor. LINKX and MixHop are architectures designed for heterophilious graphs, a specialization validated by their strong performances in Table 4.2 on those datasets. However, GOAT constantly achieves competitive results on *all four* datasets, which reveals the adaptive ability of the attention functions. We highlight that our goal is not to beat GNNs universally; specific priors can still be useful factors when designing architectures. Further, in accordance with the no-free-lunch theorem, winning all comparisons on datasets with diverse properties is expected to be difficult. Rather, our intention is to develop a model that can learn different inductive biases as required, adapting seamlessly to different use cases. This flexibility is an important quality in practice as practitioners may encounter datasets for which knowledge about appropriate priors is scarce. Note that we further try to compare with GPS [127], which is a powerful scalable graph transformer on graph-level tasks. However on large-scale node tasks GPS

Table 4.2: Experimental results. Note that the "Overall performance average" is the mean value of the two "Performance avgerage" values in the upper and lower tables. The darker blue marks the best result in each row, while the lighter shade marks the runner-up. Evaluation metric is prediction accuracy on the test set. Test accuracies are reported based on the hyperparameter setting yielding the highest validation accuracies. Where possible, we validate our baselines against the results in Lim et al. [2]. OOM means out of memory error through our experiments.

		General GNN		Heterophilious GNN		Transformer	Ours
Dataset	Train ratio	GCNJK	GAT	LINKX	MixHop	GPS	GOAT
`ogbn-arxiv`	official 54%	69.57 ± 0.20	71.95 ± 0.36	66.18 ± 0.33	71.29 ± 0.29	OOM	72.41 ± 0.40
`ogbn-products`	official 8%	72.84 ± 0.36	79.45 ± 0.59	71.59 ± 0.71	73.48 ± 0.29	OOM	82.00 ± 0.43
Performance average		71	76	69	72	OOM	77
`arxiv-year`	random 10%	43.34 ± 0.08	38.34 ± 0.10	46.22 ± 0.24	45.13 ± 0.25	OOM	49.44 ± 0.11
`arxiv-year`	random 20%	44.77 ± 0.10	39.19 ± 0.12	49.16 ± 0.42	47.18 ± 0.24	OOM	51.21 ± 0.44
`arxiv-year`	random 50%	47.74 ± 0.23	40.27 ± 0.20	53.53 ± 0.36	50.37 ± 0.25	OOM	53.57 ± 0.18
`snap-patents`	random 10%	32.50 ± 0.10	32.72 ± 0.10	49.74 ± 0.46	33.57 ± 0.06	OOM	44.31 ± 0.43
`snap-patents`	random 20%	32.97 ± 0.06	32.96 ± 0.09	54.32 ± 0.50	33.96 ± 0.06	OOM	49.55 ± 0.31
`snap-patents`	random 50%	33.52 ± 0.05	33.10 ± 0.09	60.12 ± 0.23	34.28 ± 0.07*	OOM	54.97 ± 0.23
Performance average		39	36	52	41	OOM	51
Overall performance average		55	56	61	57	OOM	64

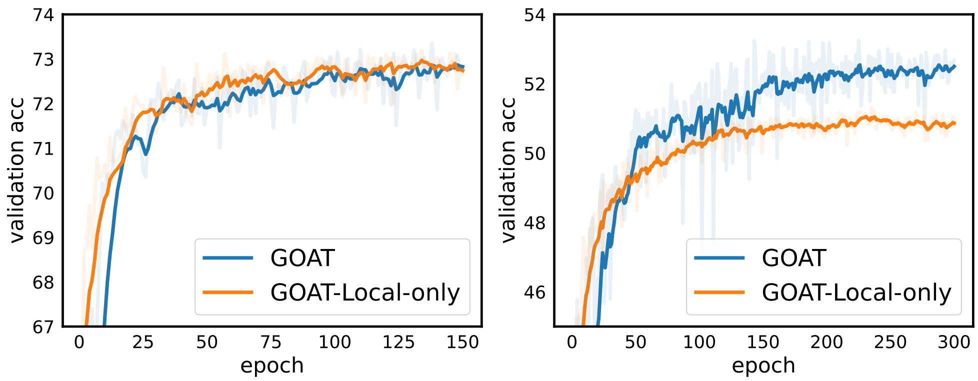
always yields OOM even on the smallest `arxiv` datasets.

4.5 Ablation Studies & Analysis

GOAT vs. GOAT-Local-only. In Figure 4.2(a) & 4.2(b) we ablate to only use the local module to analyze our architecture design. On the `ogbn-arxiv` dataset where homophily is prevalent, the local-only model is as strong as the complete GOAT. While on the `arxiv-year` dataset there shows clear discrepancy between the two, where the global module brings a salient 3% boost. The results provide a strong validatioin towards our main intuition, that the global module increases expressive power by modeling long-range interactions. And such ability is

Table 4.3: Codebook size abalation study.

codebook size	ogbn-arxiv	arxiv-year
512	71.92	50.90
$1k$	71.99	51.21
$2k$	72.14	52.51
$4k$	72.41	53.57



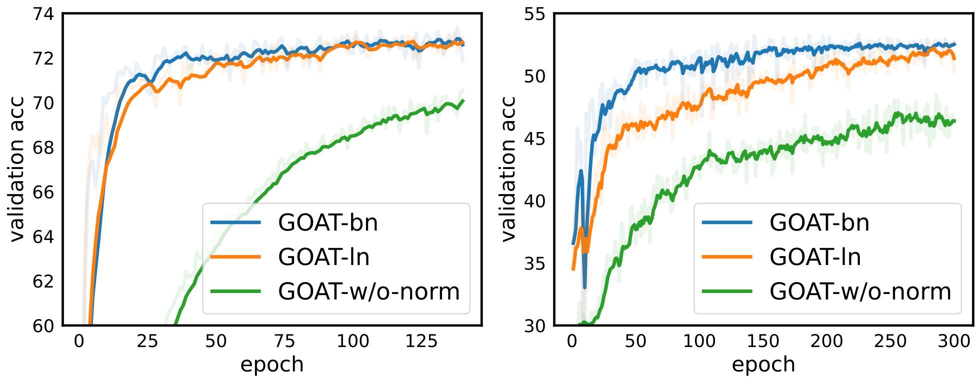
((a)) `ogbn-arxiv` ((b)) `arxiv-year`

Figure 4.2: Ablation study on the local module.

especially effective towards heterophilious graphs, which is intuitive as many other successful heterophilious GNN architectures aimed at broadening the range of message passing as well as learning from long-range interactions.

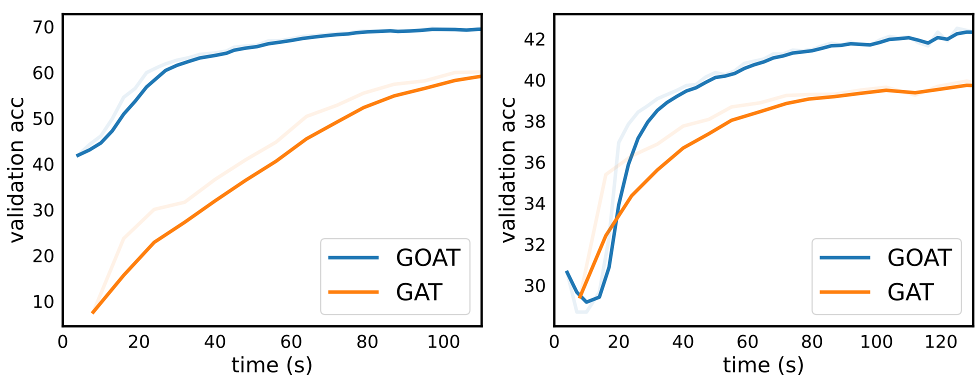
Codebook size. We run ablation studies on the codebook size for a better understanding and interpretation of the GOAT model. The codebook size refers to the k value in the K-Means algorithm. We summarize the results in Table 4.3. Note the heterophilious dataset `arxiv-year` is more sensitive to the adjustment of the codebook size, showing the efficacy of global attention module at learning long-range interactions.

Batch norm vs. Layer norm. Normalization is important to transformers. We ablate on the normalization technique selection in Figure 4.3(a) & 4.3(b). We see that layer norm can be as good as batch norm in our model but converges much slower, and evidently "no normalization" is



((a)) `ogbn-arxiv` ((b)) `arxiv-year`

Figure 4.3: Ablation study on the normalization technique.



((a)) `ogbn-arxiv` ((b)) `arxiv-year`

Figure 4.4: Convergence speed comparison.

not a good choice for our architecture.

Depth. We constantly have 1 layer of attention modules in our model, for both the global and local part. In fact it is challenging to extend our GOAT model to multi-layer attention computation like common transformers. Note that our attention modules are not strict *self-attention* modules, whose input and output are of the same length. As denoted in Fig 4.1, our input includes the predictive node, sampled neighbors, and the codebook, but only the the predictive node attends to others and gets aggregated features. So after the attention function the state of feature for sampled neighbors and the codebook is behind that of the predictive node, where future rounds of attention computation is less intuitive. A straightforward strategy to address this conundrum is to recursively sample neighbors of neighbors and add more codebooks by each layer, but apparently this will greatly adds to the overhead and renders the model less efficient. Due to computational limitation we do not carry out the experiments, but how to make the model deeper is an interesting research opportunity.

Efficiency. The neighbor sampling bottleneck is the major limitation of our method. Note that although the local module intuitively helps learn better representations, the neighbor sampling (NS) algorithm does introduce a small scalability concern. It is well-known that the neighborhood explosion problem for NS has complexity $O(d^l)$ for l-hop neighborhoods, where d is the average degree. Our sampling method in the local module is a plain adaptation of NS so that GOAT will share efficiency issues of NS. However when we only use the global module of GOAT, its convergence rate outperforms that of GAT-NS by a large margin according to Figure 4.4(a) and Figure 4.4(b) due to the linear complexity benefit. In Table 4.4 we report the absolute elapsed running time per training epoch for GAT-NS and GOAT on datasets with $1k$ and $4k$ as batch size. We see that GOAT has competitive running speed compared with GAT that is powered by NS for scalable training, while our GOAT-Global-only variant overwhelmingly outperforms the other two thanks to the linear training complexity. Detailed hyperparameter setup can be found in the Appendix.

4.6 Related Work

Scalability. The poor scalability of GNNs to large graphs remains a major obstacle to deploying GNNs for enterprise-scale applications. Scalable GNN algorithms mainly involve node-, layer-, and graph-wise sampling methods.

Table 4.4: Running time in second (s) per epoch of different models. GOAT-G refers to GOAT-Global-only model variant.

batch size	**model**	ogbn-arxiv	ogbn-products
1k	GAT	9.76	127.42
1k	GOAT-G	2.88	5.58
1k	GOAT	8.00	109.51
4k	GAT	8.31	113.47
4k	GOAT-G	2.33	4.36
4k	GOAT	7.62	101.73

Hamilton et al. [17] proposed neighbor sampling (NS) to repeatedly sample neighbors for message passing, but this approach leads to the exponential neighborhood explosion problem mentioned at the end of Section 4.3. One technique to address this issue is the layer-wise sampling proposed in Huang et al. [34]. ClusterGCN [30] and GraphSAINT [31] both perform subgraph sampling so that GNNs can be run in a "full-batch" manner but on tractable subgraphs that hopefully approximate the global graph semantics - we employ the latter of the two for training our baseline GCNJK, GAT, and MixHop models. A recent method, GAS [100], stores historical node features to help both training and inference process. Finally, it is also established that transformers suffer from inherent scalability issues. In response, a plethora of efficient transformers have been proposed that employ different techniques to improve their complexity including sparse attention maps [111, 131], clustering-based schemes [110, 132], and low-rank projections of the attention matrix [95, 133].

Graph transformers. Most successful applications of transformers to graphs problems have only considered graph classification. Graphormer [104] is the representative global transformer on small graphs with customized centrality, edge, and spatial encodings. Other strong architectures include Rong et al. [69], Dwivedi and Bresson [107], Kreuzer et al. [108], Maziarka et al. [109].

Existing graph transformers [107, 115, 117, 127] for node classification use the same recursive message passing scheme as traditional GNNs. The attention module computes attention scores as the weights for 1-hop neighborhood smoothing, similar to the attention mechanism in GATs [28]. These models fail to demonstrate the ability to learn from larger contexts and generally do not outperform traditional GNNs. An example architecture that does implement a form of global attention, GraphBERT [70], fails to solve scalabilty issues due to requiring full-batch training.

Heterophily. Recently a body of work has focused on adapting GNNs to heterophilious graphs. LINKX [2] is a simple model that coerces 2-hop (but not 1-hop) neighbors to have the similar labels. H2GCN [121] aggregates features from 1-hop and 2-hop neighbors separately. MixHop [119] aggregates information from diverse degrees of smoothed features. GPR-GNN [122] is similar to MixHop, but uses a more complex adaptive aggregation operation. GPR-GNN works well on some kinds of data but does not scale to large graphs. One closely related work is Non-local GNN [134], which does an approximate global attention leveraging attention-based sorting. The model successfully reduces time complexity from $O(n^2)$ to $O(nlog(n))$, but cannot support mini-batch training, so its scalability is limited.

While progress has certainly been made towards more effective graph transformers and models specifically suited to heterophilious graphs, to date, no graph transformer has emerged that simultaneously relaxes the restriction of a homophilious inductive bias (whilst remaining performant in that setting) and easily handles large-scale node classification tasks.

4.7 Conclusion

We propose GOAT, a global transformer that works on both homophilious and heterophilious node classification tasks. Our model makes global attention possible by dimensionality reduction and we prove our approximate approach has bounded error compared with the true global mechanism. Experiments reveal GOAT's strong performances on large-scale graphs. We hope our work can spur more research into universal graph neural architecture who can adapt to different inductive biases.

4.8 Appendix

4.8.1 More Theoretical Analysis

Note that in the demonstration process below, we assume positional information is already fused into node features.

Proposition 7. *There exists a distribution of random projection matrices $P \in \mathbb{R}^{n\times k}$ such that for any linear weight matrices $W_K, W_Q, W_V \in \mathbb{R}^{f\times d}$, node feature matrix $X \in \mathbb{R}^{n\times f}$, mini-batch of nodes $X_B \in \mathbb{R}^{b\times f}$, column vector $v \in \mathbb{R}^n$ of XW_V, and any choice of $\varepsilon > 0$,*

$$\Pr\left(\|A_B PP^\top v - A_B v\|_F < \varepsilon \|A_B v\|_F\right) > 1 - O(1/n), \tag{4.10}$$

with $A_B = \operatorname{Softmax}\left(X_B W_Q \left(XW_K\right)^\top / \sqrt{d}\right)$ and $k = O\left(\log(n)/\varepsilon^2\right)$.

Our goal here is to find a projection matrix P to project both the attention matrix A_B and features v into some low dimension space, enabling us to replace the attention matrix $A_B \in \mathbb{R}^{b\times n}$

with $\widetilde{A}_B = A_B P \in \mathbb{R}^{b \times k}$ and $v \in \mathbb{R}^n$ with $\widetilde{v} = P^T v \in \mathbb{R}^k$. Despite the existence of such P matrix, we argue that it is impractical to apply Proposition 7 in practice. The reason is that it is hard to materialize $\widetilde{A}_B$ in memory so the explicit computation of A_B is inevitable, which does not help improve scalability as calculating all attention matrices batch-by-batch is still $O(n^2)$. Below, we describe a route to escape this problem.

4.8.2 More Ablation Studies

Local range. To show the efficacy of our local module, we further run the ablation study to have nodes attend to 1-, 2-, 3-hop neighborhoods within the local transformer of GOAT. Results are included in Table 4.5. We can see that the local module plays an important role in extracting features from local fields and contributes to the final predictions of the GOAT model. Note that arxiv-year and snap-patents have 50% training samples.

Table 4.5: Ablation study on the hop range of GOAT local module.

	ogbn-arxiv	**arxiv-year (50%)**	**ogbn-products**	**snap-patents (50%)**
GOAT-local-1-hop	71.23	50.18	78.18	47.47
GOAT-local-2-hop	72.26	52.06	78.74	49.44
GOAT-local-3-hop	**72.41**	**53.57**	**82.00**	**54.97**

4.8.3 Proofs

This section lists theoretical proofs of our propositions and theorems.

4.8.3.1 Proof of Proposition 7

Proof. To help build the proof, we utilize the Johnson–Lindenstrauss (JL) Lemma from Johnson and Lindenstrauss [78], Kane and Nelson [79].

Lemma 2. *For any integer $d > 0$, and any $0 < \varepsilon$, $\delta < 1/2$, there exists a probability distribution on $k \times d$ real matrices for $k = \Theta(\varepsilon^{-2} \log(1/\delta))$ such that for any $x \in \mathbb{R}^d$,*

$$\mathop{\mathbb{P}}_{P}((1-\varepsilon)\|x\|_2 \leq \|Px\|_2 \leq (1+\varepsilon)\|x\|_2) > 1 - \delta. \tag{4.11}$$

Using Lemma 2 and Boole's inequality, for any $x, y \in \mathbb{R}^{1 \times n}$, we have

$$\Pr(\|xPP^\top y^\top - xy^\top\|_F \leq \varepsilon \|xy^\top\|_F) > 1 - 2\delta.$$

Now we choose to swap x with any row vector a out of A_B and y with v so we have

$$\Pr(\|aPP^\top v^\top - av^\top\|_F \leq \varepsilon \|av^\top\|_F) > 1 - 2\delta.$$

Leveraging Boole's inequality, we get

$$\Pr(\|A_B PP^\top v^\top - A_B v^\top\|_F \leq \varepsilon \|A_B v^\top\|_F) > 1 - 2b\delta > 1 - 2n\delta.$$

By choosing δ to be $O(1/n^2)$ we finally get

$$\Pr(\|A_B PP^\top v^\top - A_B v^\top\|_F \leq \varepsilon \|A_B v^\top\|_F) > 1 - O(1/n),$$

with $k = O\left(\log(n)/\varepsilon^2\right)$.

□

4.8.3.2 Proof of Theorem 4

Proof. We follow the techniques in Wang et al. [95] to provide the proof. We define $P_A = \delta P$ and $P_V = e^{-\delta} P$, δ is a constant. We aim to prove

$$\Pr\left(\left\|\exp\left(SP\right)P^\top X_V - \exp(S)X_V\right\|_F \leq \varepsilon \|\exp(S)\|_F \left\|X_V\right\|_F\right) > 1 - O(1/n), \tag{4.12}$$

where $X_V = XW_V$ for simplicity. Using triangle inequality, we have

$$\begin{aligned}\left\|\exp\left(SP\right)P^\top X_V - \exp(S)X_V\right\|_F \leq & \underbrace{\left\|\exp(S)PP^\top X_V - \exp(S)X_V\right\|_F}_{a} \\ & + \underbrace{\left\|\exp\left(SP\right)P^\top X_V - \exp(S)PP^\top X_V\right\|_F}_{b}.\end{aligned}$$

For part a, based on Proposition 1, we have

$$a \leq \varepsilon \|\exp(S)\|_F \left\|X_V\right\|_F.$$

For part b, we have

$$b \leq \left\|\exp\left(SP\right) - \exp(S)P\right\|_F \|P^\top X_V\|_F.$$

Given that the exponential function is Lipschitz continuous in a compact region, and also based on the Equation 2, we have

$$b \leq o(\|\exp(S)\|_F \|X_V\|_F).$$

Based on part a and part b, we finally attain

$$\Pr\left(\left\|\exp\left(SP\right) P^\top X_V - \exp(S) X_V\right\|_F \leq \varepsilon \|\exp(S)\|_F \left\|X_V\right\|_F\right) > 1 - O(1/n).$$

□

4.8.3.3 Proof of Theorem 5

Proof. This proof is built upon the usage of Lipschitz constant and quantization error.

$$\begin{aligned}
\left\|\widetilde{X}_B^{\text{out}} - X_B^{\text{out}}\right\|_F &= \|\widetilde{A}_B \widetilde{X} W_V - A_B X W_V\|_F \\
&= \|f_W(\widetilde{X})\widetilde{X} W_V - f_W(X) X W_V\|_F \\
&\leq \underbrace{\|f_W(\widetilde{X})\widetilde{X} - f_W(X) X\|_F}_{a} \|W_V\|_F,
\end{aligned}$$

where for the part a, we have

$$\begin{aligned}
a &= \|f_W(\widetilde{X})\widetilde{X} - f_W(X)\widetilde{X} + f_W(X)\widetilde{X} - f_W(X) X\|_F \\
&\leq \|f_W(\widetilde{X}) - f_W(X)\|_F \|\widetilde{X}\|_F + \|f_W(X)\|_F \|\widetilde{X} - X\|_F.
\end{aligned}$$

Note that

$$\|\widetilde{X}\|_F \leq \|P\text{diag}^{-1}(\mathbf{1}_n P)P^\top\|_F \|X\|_F \leq \sqrt{k}\|X\|_F,$$

$$\|X - \widetilde{X}\|_F \leq \varepsilon \|X\|_F,$$

$$\|f_W(\widetilde{X}) - f_W(X)\|_F \leq \operatorname{lip}(f_W) \|X - \widetilde{X}\|_F.$$

When we assume $\|X\|_F = O(\|A_B\|_F)$, we have

$$\left\|\widetilde{X}_B^{\text{out}} - X_B^{\text{out}}\right\|_F \leq \varepsilon \cdot [1 + O(\operatorname{lip}(f_W))] \|A_B\|_F \cdot \|X\|_F \cdot \|W_V\|_F.$$

□

4.8.3.4 Proof of Corollary 6

Proof.

$$\widetilde{X}_B^{\text{out}} = \text{Softmax}\left(X_B W_Q (\widetilde{X} W_K)^\top / \sqrt{d}\right) \widetilde{X} W_V,$$

$$\widetilde{X}_B^{\text{out}} = \text{Softmax}\left(X_B W_Q \left(P \operatorname{diag}^{-1}(\mathbf{1}_n P) P^\top X W_K\right)^\top / \sqrt{d}\right) P \operatorname{diag}^{-1}(\mathbf{1}_n P) P^\top X W_V,$$

$$\widetilde{X}_B^{\text{out}} = \text{Softmax}\left(X_B W_Q \left(P \widetilde{K}\right)^\top / \sqrt{d}\right) P \widetilde{V},$$

$$\widetilde{X}_B^{\text{out}} = \text{Softmax}\left(X_B W_Q \widetilde{K}^\top P^\top / \sqrt{d}\right) P \widetilde{V},$$

note that the row vector of $P \in \mathbb{R}^{n \times k}$ is one-hot, so $X_B W_Q \widetilde{K}$ times $P^\top$ works as copying the attention scores according to the assignment of centroid of each node. In the same way, $\text{Softmax}(\cdot)P$ aggregates the Softmax logits based on the assignment through summation.

Say a random row vector x from X_B, $x W_Q \widetilde{K}^\top / \sqrt{d}$ is denoted as a row vector $[s_1, s_2, \ldots, s_k] \in \mathbb{R}^{1 \times k}$, $(\mathbf{1}_n P)$ is denoted as $[n_1, n_2, \ldots, n_k]$, then we can see that

$$\left(\text{Softmax}\left(x W_Q \widetilde{K}^\top P^\top / \sqrt{d}\right) P\right)[i] = \frac{n_i \exp(s_i)}{\sum_{j=1}^{k} n_j \exp(s_j)},$$

so we can cancel the P both inside and outside Softmax by adding a weighting term to make the exponential computation weighted with the centroid size, where we attain

$$\widetilde{X}_B^{\text{out}} = \text{Softmax}\left(X_B W_Q \widetilde{K}^\top / \sqrt{d} + \log\left(\mathbf{1}_n P\right)\right) \widetilde{V}.$$

$\square$

4.8.4 Experimental Details

This section describes in more detail the experimental setup for the empirical results presented in Section 4.

4.8.4.1 Dataset Downloading

We refer readers to Hu et al. [12] and Lim et al. [2] and their official repos for dataset downloading.

4.8.4.2 GOAT

For the hyperparameter selections of our GOAT model, besides what we have covered in the setup part of the experiment section that datasets share in common, we list other settings in Table 4.6. Each experiment is repeated four times to get the mean value and error bar. We use Adam optimizer with lr 1e-3.

Table 4.6: Proposed Model Dataset-Specific Hyperparameter Settings

Model	Dataset	# Heads	# Hidden Channels
GOAT	`ogbn-arxiv`	4	128
	`ogbn-products`	2	256
	`arxiv-year`	4	128
	`snap-patents`	2	128

4.8.4.3 Baseline Models

Since the heterophilius datasets on which we benchmark the GOAT model are derived from Lim et al. [2], in order to facilitate as fair a comparison as possible, especially against the LINKX model proposed in the same work, we utilize their implementation provided at the official repo[1]. This library also provides reference implementations of other GNN architectures, and we utilize those as well. Following the procedure described both in the paper and implicitly

[1]https://github.com/CUAI/Non-Homophily-Large-Scale

by their codebase, we run a hyperaparameter sweep for each model on each dataset. For each combination of parameters, 5 models are trained using different initialization seeds to determine a mean value and error bars, and then the final hyperparameters are selected based on accuracy on the validation set. These final parameters correspond to the settings used to realize the "official train split" numbers in Table 2 in the main work for `ogbn-arxiv` and `ogbn-products` and the 50% train split numbers for `arxiv-year` and `snap-patents`. These same parameters are also used when performing the sample complexity experiments with train splits of 10% and 20% for the latter two datasets.

For the baseline GNNs we chose a Graph Convolutional Network with Jumping Knowledge (GCNJK) and a Graph Attention Network (GAT), and for a second heterophily-specific model to complement LINKX, we consider MixHop. As described in Section 4 of the main work, we use ERM to train all models including the baselines. We also focus on the minibatch setting rather than "full-batch" training as a primary feature of the GOAT model is its native scalability through minibatch training. The batching algorithm we use for the GCNJK, GAT, and MixHop models is the GraphSAINT Random Walk based sampler (LINKX uses its own adjacency row-wise sampling scheme, see Lim et al. [2] for details). In the spirit of fair evaluation, we make the model and sampler choices based on the fact that according to Lim et al. [2], each of these are the most performant two models in the homophily and heterophily-specific design categories on the datasets under evaluation.

Hyperparameter Settings: For all four baseline models we tune two main parameters: the number of layers, and the number of hidden channels (dimension) of each layer. We also evaluate two subgraph sizes, or batch sizes, for the SAINT sampler (number of roots used for the random walk). For the GAT model only, we also tune the number of attention heads. We evaluate the same

Table 4.7: Baseline Model Dataset-Specific Hyperparameter Settings

Model	Dataset	# Layers	# Hidden Channels	SAINT Batch Size
GAT	`ogbn-arxiv`	2	32	10,000
	`ogbn-products` *	–	–	–
	`arxiv-year`	2	32	10,000
	`snap-patents`	2	32	10,000
GCNJK	`ogbn-arxiv`	2	128	10,000
	`ogbn-products`	4	256	5,000
	`arxiv-year`	4	256	10,000
	`snap-patents`	2	256	10,000
MixHop	`ogbn-arxiv`	2	128	10,000
	`ogbn-products`	3	128	5,000
	`arxiv-year`	4	128	10,000
	`snap-patents`	2	128	10,000
LINKX	`ogbn-arxiv`	1	64	N/A
	`ogbn-products`	1	128	
	`arxiv-year`	1	256	
	`snap-patents`	1	16	

parameter ranges described in Lim et al. [2] and defined by their codebase, and simply report the final parameters selected in Table 4.7. Selected parameters that are shared amongst all model and dataset pairs include training for 500 epochs, using the AdamW optimizer with a learning rate of 0.01, and creating 5 SAINT subgraphs per epoch. Model specific selected parameters include using 8 heads for the GAT, concatenative jumping knowledge for the GCNJK model, and the 2 hop setting for MixHop. For all settings that we choose as final, if possible, we verify that the accuracy is within $\pm 1\%$ of the value reported in Lim et al. [2] as "best" for each model on each dataset.

There are two details of particular note concerning Table 4.7 (and corresponding results in

Table 2 in the main work). First, for the `snap-patents` dataset we do not use the ClusterGCN sampler for the MixHop architecture as it proved too time and memory intensive for our hardware. We ground this choice in the fact that according to the performance reported in Lim et al. [2], the performance of MixHop with cluster sampling would still have been below the performance of GOAT by approximately 8 accuracy points. As an additional comment, the computational costliness of this method was also a challenge in their work, precluding it from parts of their evaluation. In the spirit of scalability, overall, we see GraphSAINT as being a more relevant choice for our comparison due to its more favorable scaling characteristics.

Second, also in service of a competitive evaluation, we chose to report performance of the GAT model on `ogbn-products` * pulled from the Open Graph Benchmark's official leaderboard[2] for this dataset since the minibatch performance we achieved with the SAINT sampler was significantly lower than the reference result (as well as that of our GOAT model). The result from the leaderboard was trained using the neighbor sampling algorithm.

For Fig 4.4(a) and Fig 4.4(b), we set batch size as 10K. We use 4K as codebook size. GAT leverages NS [20,10,5] to do minibatching. The total running time involve evaluation time.

[2]https://ogb.stanford.edu/docs/leader_nodeprop/

Chapter 5: OpenTab: Advancing Large Language Models as Open-domain Table Reasoners

5.1 Introduction

The field of natural language processing has rapidly advanced in tasks of text generation [135] and knowledge reasoning [136], driven by general-purpose large-scale generative models like GPT4 [35]. However, these generative models have inherent drawbacks, like hallucinations [137] and limited specialized knowledge. They cannot be used to perform tasks involving data that has not been trained on, like answering questions about recent events or about proprietary enterprise data. Recent advancements in retrieval-based methods [138] that aim to equip generative models with expanded information, allows for grounded responses based on real-time or proprietary data.

Many retrieval-based LLMs primarily target textual data from the web or text corpora, overlooking the wealth of information stored in structured tables. When applied to the tables, retrieval-based LLMs encounter the following challenges: (1) Structured tables have diverse data types, notably numerical data presented in large or precise numbers, which results in a high token usage and potentially leads to memory and computational constraints for the model. (2) LLMs, mainly those optimized for natural language understanding, fail to comprehend the complex relationships within tabular data for effective data transformation and answer extraction. (3) The restricted maximum context length of LLMs poses difficulties in managing a multitude of retrieved tables of varying sizes, particularly when dealing with tables containing millions of rows.

This work aims to develop an LLM-based framework for the open-domain table reasoning task [139, 140]. Here, we concentrate on two specific tasks: table-based question answering (QA) and fact verification. For both tasks, the framework is presented with natural language queries and a corpus of tables. In the open-domain setting, the gold tables, which serve as the evidence for queries to generate responses, are not provided. Therefore, the challenge lies in automatically identifying and retrieving pertinent knowledge from the table corpus and also formulating a coherent natural language response. A task example of table QA is in Figure 5.1.

Existing LLM-based table reasoning methods, like BINDER [141] and DATER [142], operate in the closed-domain setting, in which table retrieval is not required and a pre-defined gold table is available for each query during test time. While this assumption simplifies the reasoning process, it does not hold in most real-world scenarios as manually providing gold grounding information is costly and impractical. In the open-domain context, predictions must be grounded in a selection of relevant tables, and the system should possess the reasoning capabilities to sift through incorrect tables [143] to derive the correct output. Meanwhile, existing open-domain table reasoning systems [144, 145, 146] have a fine-tuned encoder for retrieval and a fine-tuned table reasoner [147] for QA as well, which necessitates specialized training for specific tasks and is less flexible. Moreover, although they perform well on observed data after fine-tuning, the performance drops on new tables due to constrained transferability.

We propose OPENTAB, an open-domain & end-to-end table reasoning framework. As illustrated in Figure 5.1, OPENTAB leverages a RETRIEVER to fetch relevant tables, generates

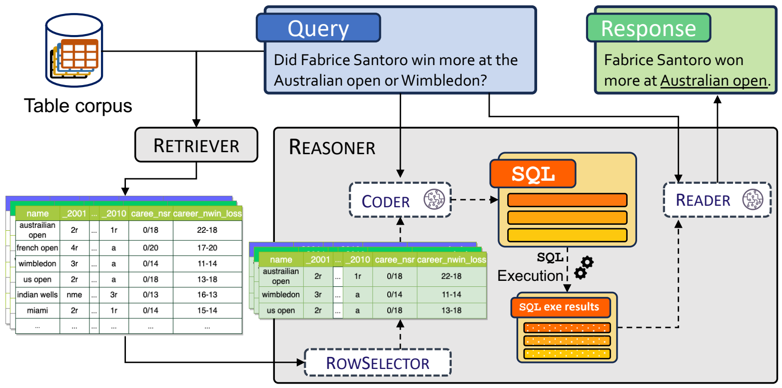


Figure 5.1: An overview of OPENTAB pipeline. OPENTAB uses a RETRIEVER to retrieve relevant sampled tables from a corpus of tables for a given natural language query, and then use a REASONER to output a natural language response.

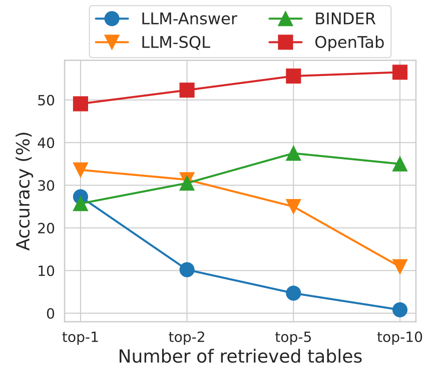


Figure 5.2: The accuracy with an increasing numbers of retrieved tables. OPENTAB has a consistent increase in accuracy with more tables.

programs from a CODER as intermediary reasoning steps, and delegates the final solution to a READER. Following thorough empirical assessment, we opt to implement the BM25 [148] algorithm as the table RETRIEVER because of its scalability and effectiveness. To mitigate challenges of LLMs in processing and understanding structured tables, we utilize an LLM as a CODER to generate high-quality SQL programs for efficient table parsing. Moreover, an LLM-based READER works to formulate the final response based on the SQL execution results. By breaking down the complex reasoning job into programmatic steps, we achieve enhanced accuracy and reliability in open-domain table reasoning. Notably, the CODER and the READER modules are guided by few-shot prompting, without relying on training or fine-tuning. Our approach has several key advantages:

High Accuracy. Open-domain table reasoning presents a challenge due to the trade-off between retrieval recall and precision. Higher recall/coverage requires retrieving more tables, but this might introduce noise from less pertinent tables and therefore reduces precision and prediction accuracy. Specifically, we propose a reranking strategy called **Generative Reranking & Sequential Reasoning** to prioritize the tables with higher similarity between the natural language query and the corresponding generated SQL programs to address the trade-off, which enhances prediction accuracy as shown in Figure 5.2.

Scalability. Our approach can efficiently handle large quantities and sizes of tables. The framework is capable of generating accurate responses with access to only a limited number of rows from each table effectively extracted by ROWSELECTOR.

Robustness. We propose *simple-to-complex* prompting, a flexible and robust progressive program generation and execution strategy. By sequentially generating SQL programs with increasing levels of complexity, starting from basic column selection to advanced operations like aggregation and text operations, the model explores a wider range of possible solutions. This approach offers more adaptability in finding the optimal SQL query for a given task, and it also serves as a structured reasoning process for generating effective SQL code. Additionally, it allows for easy fallback to simpler queries if the more complex ones fail, enhancing the overall robustness of the system.

Effectiveness. We provide extensive experimental results showing that our method registers competitive performances without any fine-tuning on the target dataset. We show that OPENTAB significantly outperforms baselines in both open- and close-domain settings. For example OPENTAB outperforms the best baseline with a remarkable 21.5% higher accuracy under the open-domain scenario. We further run detailed ablation studies to validate the efficacy of our proposed designs.

5.2 Definition of Open-Domain Table Reasoning

We formally define the open-domain table reasoning task as follows: Given a natural language input query q and a database with tables T, the task is to generate a response a based on multiple retrieved tables, denoting as $\widehat{T}_q \subset T$, while the gold table set for query q is denoted as T_q. The form of the (q, a) pair is determined by the corresponding downstream task. In this work, we focus on the following two tasks:

- **Question answering**, where q is a natural language question and a is a natural language answer. E.g., q: "Which team won the NBA championship in 2022?", a: "Golden State Warriors."
- **Fact verification**, where q is a statement and a is a judgement given T, like entailment or contradiction. E.g., q: "In àlex corretja career there be no game in may 1998", a: "Entailed".

We approach the task as modeling $P(a|q, T)$. Technically, as T can be a large collection of tables, it is intractable to directly solve $P(a|q, T)$. Thus, we follow the two-step approach that is often adopted in natural language retrieval augmented models [149]. Namely, given a query q, we firstly retrieve a set of evidence tables $\widehat{T}_q \subset T$, and subsequently model $P(a|q, \widehat{T}_q)$ in place of $P(a|q, T)$. In this work, we expect to leverage LLMs as the backbone for modeling given their capability in few-shot inference and reasoning [150].

Note that the key difference between open-domain and closed-domain table reasoning task is that in the closed-domain setting the tables T_q are explicitly provided for each query q. In contrast, in the open-domain setting, there is possibility that $T_q \cap \widehat{T}_q = \emptyset$ depending on the ability of the retriever.

5.3 Method

Overview. We present an overview of OPENTAB in Figure 5.1. To enable our pipeline to automatically handle large-scale tabular data in terms of both table size (number of rows and columns) and table quantity, we divide the overall system into two components: a RETRIEVER (in Section 5.3.1) and a REASONER (in Section 5.3.2). For RETRIEVER, we advocate the BM25 for table retrieval tasks as it provides scalable and competitive retrieval performance. For the REASONER, we leverage the CODER powered by LLM to generate SQL queries based on retrieved tables T_q. The final response a is then extracted by an LLM-based READER module to ensure the accuracy, efficiency, and robustness of the REASONER against generation stochasticity. In Section 5.3.3, we introduce a Generative Reranking & Sequential Reasoning (GRSR) strategy for open-domain reasoning, which sequentially generates SQLs for a set of retrieved tables and then reranks them based on query similarity, effectively addressing the hallucination issues.

5.3.1 Table Retriever

Effective table retrieval in the open-domain setting [140, 151] remains an open question [152]. We select BM25 [1] [153] as the table retriever in our framework for the following considerations. First, BM25, a probabilistic-based ranking function used in information retrieval systems to rank documents, is a standard sparse retrieval method that works efficiently on large corpora, considering term frequency and inverse document frequency with saturation. Second, it is easy to use without the necessary fine-tuning process of Dense Passage Retrieval (DPR) [145]. Third, empirically, we show that BM25 achieves competitive retrieval performance compared

[1]Implementation based on https://github.com/dorianbrown/rank_bm25

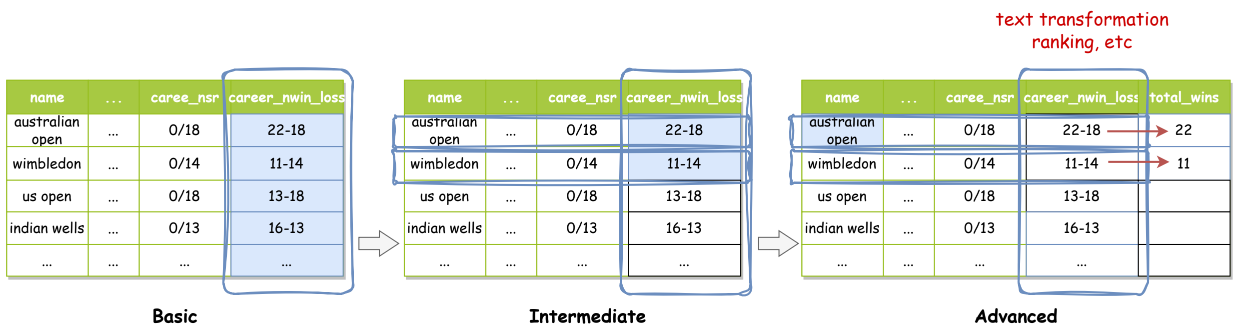


Figure 5.3: Examples illustrating the progressive *Simple-to-complex* SQL proficiency. The *basic* SQL queries mainly select specific columns. The *intermediate* incorporates both column and row selection. The *advanced* SQL employs additional operations like aggregation and text operations that can manipulate and transform the tabular data. The cells in blue are outputs of the SQL programs.

with dense methods such as DPR in Section 5.4.3. Given these, BM25 as RETRIEVER together with REASONER powered by LLMs make the OPENTAB system off-the-shelf usable without the necessity of fine-tuning on target databases, which is a significant advantage for industrial applications.

5.3.2 Table Reasoner

The REASONER processes natural language queries along with table schema and sampled rows, facilitated by a CODER and a ROWSELECTOR. The ROWSELECTOR ensures that the pertinent rows are provided to the LLM for effective processing. The CODER generates SQL programs of increasing complexity. Finally, the READER leverages an LLM to parse and extract the final response.

5.3.2.1 CODER

On the table reasoning task, LLMs (e.g., ChatGPT and Falcon) are capable of generating SQL queries that can be executed on databases to extract the final answer [154, 155, 156]. This approach is referred to as the LLM-SQL baseline. However, the generated SQLs towards direct question answering can be less robust with both syntax and semantic errors during execution when facing complex reasoning tasks [141], which might require multiple operations such as aggregation, comparison, and sorting [140].

In CODER, we propose a new ***simple-to-complex*** prompting strategy for effective SQL generation. Specifically, for each input query, we prompt the LLM to sequentially generate three SQL programs with ascending complexities and functionalities, which are:

- *SQL-basic*: focusing on column selection, which sets the groundwork for understanding how to fetch specific data from a database.
- *SQL-intermediate*: incorporating both column and row selection. This means extracting particular columns and filtering rows based on specified criteria, enhancing precision in data gathering.
- *SQL-advanced*: empowering additional operations including but not limited to aggregation functions and text operations. Aggregation empowers data summarization, while text operations facilitate the manipulation and transformation of string data.

Figure 5.3 shows the high-level idea of this *simple-to-complex* prompting strategy. The reason for such a design is that generating solid SQLs can be challenging for complex input queries, so we incrementally generate SQLs in reasoning steps from simple to complex. Moreover, with the READER module (explained later), the generated SQL programs do not necessarily need to produce the final answer directly. This is infeasible when the input query is not solvable by pure SQL programs [141]. Therefore, *SQL-basic* and *SQL-intermediate* can not only function as reasoning steps towards the ultimate solid *SQL-advanced* but can also lead to a final correct response. This provides more flexibility and robustness than generating a single SQL. After the generations of three SQL programs, we first test *SQL-advanced* by execution as verification. We proceed to the READER module for final response extraction if it returns valid non-empty results. If the SQL fails the verification, we turn to verify simpler ones until all SQLs are exhausted. Figure 5.5 displays the prompt and generation structure of CODER. For concrete problem solving example, see Figure 5.6.

5.3.2.2 READER

To expand the limited capability of SQL in solving natural input queries, we use a READER module that leverages LLM to digest the intermediate SQL execution results and formulate the final response. Rather than providing only the execution results, we further supply the READER with broader context from CODER, including table schema, sampled rows, and generated SQL query. This enables the READER to better understand the contextual background and semantics needed for accurate predictions. We opt to provide READER with these additional contexts based on analysis showing poor performance when only giving the execution results without exposure to the richer contextual information from CODER (see ablation studies). As shown in Figure 5.5, our framework prompts the READER with this broader context encapsulating the entire SQL generation process.

5.3.2.3 ROWSELECTOR

Besides table schema, table contents are also critical for effective SQL generation. Ideally, the entire table should be input to the LLM for the complete view. However, LLMs can only handle tables with large sizes within their token capacity limits. In order to balance scalability and reasoning ability, we propose to use the RowSelector to harness a few rows that are most relevant to the input q to be placed into the prompt of LLMs. Technically, we leverage BM25 to rerank and choose the top-k rows based on relevance to q.

5.3.3 Open-domain Table Reasoning

In contrast to a closed-domain setting where the table used for reasoning is provided, open-domain reasoning operates over a much larger and unconstrained table store, causing an inherent trade-off between precision and recall of the retrieved tables. More tables need to be retrieved to achieve higher coverage (i.e., recall values), which inevitably brings irrelevant information and, thus, lower precision. Therefore, precisely identifying the correct table from the retrieved set is crucial for the pipeline to achieve accurate reasoning performance over the end-to-end open-domain task.

We propose a novel **Generative Reranking & Sequential Reasoning (GRSR)** strategy to address this pain point. Specifically, given a query q and $\widehat{\ }_q$ of k tables fetched by Retriever, SQLs are sequentially generated for each table using Coder. We then rerank the tables based on the similarity between q and the generated SQL computed using *pretrained* Cross Encoder transformers [157]. [2] The tables with the highest resultant similarity scores computed by corresponding SQLs are then selected for downstream predictions. GRSR is effective because of its ability to combat the hallucination tendency of LLMs. Ideally, Coder should only be able to generate valid SQL programs if gold set $_q$ is retrieved, such that the execution result of the SQL can be used to locate the gold tables. However, in practice, we find that Coder is likely to generate valid SQL

[2]Huggingface checkpoint with model name `cross-encoder/ms-marco-MiniLM-L-12-v2`

that can pass the execution verification even though the retrieved table is irrelevant from the query q. This hallucination problem greatly limits the expected effectiveness of generative feedback from LLMs toward retrieved tables. Given these, GRSR evades noisy retrieved tables by detecting hallucinated SQLs using a pretrained Cross Encoder. With more tables retrieved, we expect to see higher recall value but also likely with precision degradation. GRSR effectively counters such negativity introduced by more tables retrieved while leveraging the advantage of higher recall by more precisely locating the gold table and ignoring the noisy ones.

To better reveal the efficacy of the GRSR strategy, we introduce two baseline algorithms for comparison. **Joint Reasoning** approach, where all k retrieved tables are aggregated into the prompt of CODER for SQL generation and prediction simultaneously. **Sequential Reasoning** only, which is a basic sequential approach that verifies the validity of SQLs in the order defined by the retriever. If we reach a valid SQL without an execution error, we select the corresponding SQL. By default, we leverage GRSR in OPENTAB. In the ablation section, we show the performance gain of GRSR over Sequential Reasoning and Joint Reasoning, justifying the efficacy of GRSR in mitigating the hallucination problem and locating the gold table.

5.4 Experiments

5.4.1 Experiment Setup

Datasets. To evaluate the proposed approach, we use **Open-WikiTable** [140], **WikiTable-Questions** [158], and **FEVEROUS** [159] datasets. Open-WikiTable is an open-domain table question-answering dataset, where the specific table related to a given question is not provided and must be retrieved. Experiments were limited by budget to 2,000 random samples from the

Table 5.1: The accuracy on two open-domain table reasoning tasks using two different backbone LLMs. The results are presented at the top-1/2/5/10 retrieved tables.

Method	`gpt-3.5-turbo` Top-1	Top-2	Top-5	Top-10	`falcon-180B` Top-1	Top-2	Top-5	Top-10
	Open-WikiTables				*Open-WikiTables*			
LLM-Answer	0.273	0.102	0.047	0.008	0.203	0.047	0.016	0.008
LLM-SQL	0.336	0.313	0.250	0.109	0.195	0.094	0.023	0.008
BINDER	0.234	0.305	0.375	0.375	0.164	0.188	0.227	0.227
OPENTAB	**0.491**	**0.523**	**0.556**	**0.565**	**0.375**	**0.391**	**0.437**	**0.437**
	FEVEROUS				*FEVEROUS*			
LLM-Answer	0.672	**0.508**	0.492	0.453	0.680	0.476	0.375	0.289
LLM-SQL	0.609	0.460	0.515	0.422	0.328	0.242	0.195	0.094
BINDER	0.375	0.383	0.391	0.391	0.219	0.266	0.305	0.281
OPENTAB	**0.695**	**0.508**	**0.563**	**0.508**	**0.742**	**0.515**	**0.555**	**0.484**

validation set. The corpus contains 24,680 candidate tables. In contrast, WikiTableQuestions, also a table-based QA dataset, provides the relevant table containing the necessary information, making it closed-domain. FEVEROUS, on the other hand, is a fact-verification dataset that requires the identification of multiple relevant tables for reasoning and verifying factual questions. We adapted the dataset for open-domain table reasoning by filtering 323 examples from the validation set that rely solely on table data, using a corpus of 26,177 candidate tables from the FEVEROUS dataset. In line with BINDER [141], we use execution accuracy (EA) as our primary metric for evaluating performance on the Open-WikiTable and WikiTableQuestions datasets. For FEVEROUS we adopt a metric that demands the pipeline not only to make the correct predictions but also to correctly identify the gold evidence table(s) for making those predictions. We report both metrics as the accuracy term in the following descriptions. See details in Appendix 5.9.3.

Baselines. For table retrieval experiments, we conduct experiments to compare the performances of BM25 and DPR, where DPR uses BERT models[3] both with and without being fine-tuned on the target dataset. In the context of the table reasoning task, we compare with BINDER [141], an end-to-end table QA model that generates specialized symbolic languages to be executed on the table database. As the original LLM model used in BINDER (`OpenAI Codex`) is no longer available, we reproduce BINDER's results using `gpt-3.5-turbo` with BINDER's official implementation. The official implementation of BINDER does not support the open-domain scenario; and the database interface cannot support identification of multiple relevant tables. We leverage the straightforward Sequential Reasoning (SR) strategy to reduce the multi-table scenario to single-table to make BINDER applicable in the open-domain setting. Furthermore, we implement two additional baselines: (1) LLM-Answer, to prompt LLM to directly output the text-format answer based on the table and question; (2) LLM-SQL, to let LLM respond with SQL programs whose execution result will be the output. If not specified, the LLM backbone used in this work is `gpt-3.5-turbo` (default 4k-token version), and the in-context learning examples are 2-shot.

5.4.2 Main Results

Open Domain. For this setting, we first retrieve top-k relevant tables $\widehat{}_q$ from the database using the same BM25 algorithm. We run extensive experiments with 2 different LLM backbones (`gpt-3.5-turbo` & `falcon-180b`) on two datasets (Open-WikiTables & FEVEROUS). Table 5.1 summarizes the experimental results. From Table 5.1, we can see that OPENTAB significantly outperforms baselines on the Open-WikiTables dataset. For example with `gpt-3.5-turbo` OPENTAB outperforms the best baseline with a remarkable 21.5% higher accuracy under the

[3]Huggingface checkpoint with model name `bert-base-uncased`

Table 5.2: The accuracy on the closed-domain table QA task (WikiTableQuestions).

	Method	Acc
Fine-tuned	T5-3B [160]	0.519
	Tapex [161]	0.604
	TaCube [162]	0.611
	OmniTab [163]	0.633
w/o Fine-tuning	LLM-Ans	0.375
	LLM-SQL	0.414
	BINDER [141]	0.428
	DATER [142]	0.453
	OPENTAB	**0.641**

Table 5.3: The accuracy results on the closed-domain table QA task (Open-WikiTable).

Method	Acc
LLM-Answer	0.378
LLM-SQL	0.579
BINDER [141]	0.421
OPENTAB	**0.710**

top-10 scenario. Meanwhile on the FEVEROUS dataset, OPENTAB can mostly win over the baselines with non-trivial improvements. Note that the baseline methods require full tables to be fed into LLM for reasoning, which may exceed the upper bound of the input token limit, rendering an invalid prediction. While OPENTAB does not suffer from scalability issues as CODER is capable of generating high-quality SQLs with three selected representative rows. Moreover, to reproduce BINDER's performance, we follow Cheng et al. [141] to use 14-shot examples when doing in-context learning. For BINDER, we also deploy `gpt-3.5-turbo-16k` to fit the long input sequences. This group of experiments shows that OPENTAB is the more capable system at dealing with the challenging open-domain table reasoning tasks.

Closed Domain. To complement the empirical investigation, we further conduct experiments under the closed-domain scenario, where the golden evidence table is directly given without the retrieval requirements. We show the performances in Table 5.3 on the Open-WikiQuestion dataset. Similarly, to make BINDER applicable, we use `gpt-3.5-turbo-16k` with its official 14-shot examples. We can see that OPENTAB has a non-marginal 13% improvements over the baselines, further validating the efficacy of OPENTAB designs. We further evaluated the WikiTableQuestions

Table 5.4: Recall@k scores for table retrieval on the open-domain Open-WikiTables and FEVEROUS datasets. BM25 wins among the non fine-tuned methods.

Setting	Method	Recall@5	Recall@10	Recall@20	Recall@50
Open-WikiTables					
w/o Fine-tuning	BM25	**0.832**	**0.893**	**0.940**	**0.972**
	BM25*	0.422	0.489	0.561	-
	DPR-BERT	0	0	0	0
Fine-tuned	DPR-BERT*	**0.895**	**0.950**	0.973	**-**
	DPR-BERT	0.870	0.933	**0.975**	0.990
FEVEROUS					
w/o Fine-tuning	BM25	**0.919**	**0.942**	**0.956**	**0.973**
	DPR-BERT	0	0	0.001	0.003
Fine-tuned	DPR-BERT	0.698	0.767	0.831	0.902

[1] Results of *methods are from Kweon et al. [140]. Others are implemented in this work.
[2] The non fine-tuned DPR scores are approximate to 0 on both datasets due to poor transferability.

dataset [158], which is intrinsically a closed-domain table QA dataset. Results are shown in Table 5.2. It reveals the competitiveness of OPENTAB of even outperforming fine-tuned methods.

5.4.3 Table retrieval results

We report the table retrieval performance in Table 5.4. Results on the Open-WikiTable dataset are from both Kweon et al. [140] and our implementations. BM25 has competitive performances compared to the DPR, which has been fine-tuned on the labeled training dataset. On the FEVEROUS dataset, BM25 even outperforms fine-tuned BERT, which supports our intuition that sparse retrievers are simple but powerful table retrievers.

Table 5.5: Ablation studies on different proposed modules in OPENTAB. On the closed-domain Open-WikiTables dataset. STC stands for the *Simple-to-Complex* prompting SQL generation strategy.

Method	Accurate
OPENTAB	**0.710**
OPENTAB w/o STC	0.539
OPENTAB w/o STC+ROWSELECTOR	0.515
OPENTAB w/o STC+ROWSELECTOR +*broad-context* READER	0.421

5.5 Ablation Studies & Analysis

Ablation studies on the proposed modules of OPENTAB. In Table 5.5, we show the effects of designed components our OPENTAB. The evaluations happen under the closed-domain scenario. We can see that, the performance drops drastically from 0.71 to 0.54 due to the removal of the STC module, highlighting its significance. Moreover with the ablation of ROWSELECTOR, the performance continues to drop. Lastly, by disabling the longer-context functionality of READER, the performance reaches the lowest, showing the significance of incorporating detailed context information including that of the SQL generation process. For clarification, by disabling the longer-context functionality, we do not forward the table schemas and generated SQLs into READER.

Ablation studies on *Simple-to-Complex* SQLs. We test the individual efficacy of *SQL-basic*, *SQL-intermediate*, and *SQL-advanced*. Instead of moving onto the next in order if one program fails to provide valid result, we focus on one SQL program at a time for evaluation individually. Table 5.6 show the results. We can see that our design of synergizing all three kinds of design has the best performance compared with merely using solo SQLs. This is because in our *Simple-to-Complex* strategy, the solidness of the generated SQL programs will be verified on the fly, thus we can try to evade invalid SQLs that will either have syntax errors or empty execution

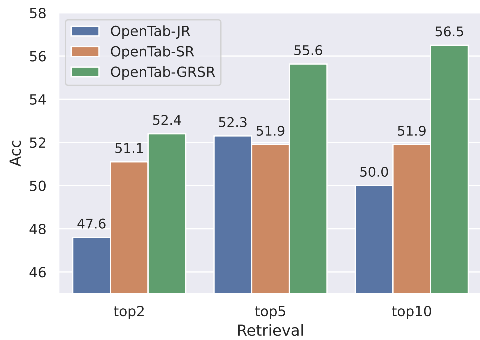


Figure 5.4: Ablation study on the open-domain strategy. "JR" stands for "Joint Reasoning" and "SR" stands for "Sequential Reasoning".

Table 5.6: Ablation studies on the STC SQL generations.

	Closed-domain	Open-domain-top2
OPENTAB	**0.710**	**0.523**
OPENTAB-adv.	0.585	0.367
OPENTAB-int.	0.609	0.406
OPENTAB-basic	0.617	0.390

results, which is common for a solo SQL generation.

Open-domain strategy. We verify the efficacy of the Joint Reasoning, Sequential Reasoning, and GRSR strategies. Results are summarized in Figure 5.4. From the plot we can see that the best performing model is OPENTAB with GRSR strategy, which is intuitive because such strategy implicitly leverages the generative power of the LLM towards identifying the correct table to build reasoning on. From the figure we can see that the accuracy of OPENTAB-GRSR constantly increases with more tables retrieved (higher recall value) despite the negative impact of more incorrect tables presented. The novel generative reranking design leads to more precisely grabbing the core information by mitigating the hallucination issue.

5.6 Related Works

LLMs for Table Reasoning. Recent work has explored leveraging LLMs for table reasoning tasks without task-specific fine-tuning. Chen [150] showed that LLMs like GPT-3 can perform substantially on table QA and fact verification datasets like WikiTableQuestions and TabFact with few-shot prompting. However, their capability degrades on large tables with 30+ rows. To address this limitation, Ye et al. [142] proposed using LLMs to extract relevant sub-tables and decompose complex questions into simpler sub-questions. Their DATER method improves LLM performance on large and complex table reasoning tasks. Alternatively, BINDER [141] maps the natural language questions into programing languages such as SQL and Python with API calls to invoke LM functions, improving the reasoning capability beyond the basic programming language grammar. He et al. [164] proposed a strategy that involves rethinking with retrieval to enhance LLM faithfulness by retrieving relevant knowledge using the chain-of-thought prompting, showing the effectiveness in a series of reasoning tasks, including tabular reasoning.

Table Retrieval. Wang et al. [151] investigated the necessity of designing table-specific dense retrievers with structure-related modules. The authors showed that DPR (Dense Passage Retriever) and DTR (Dense Table Retriever) have comparable performances, justifying the effectiveness of deploying DPR in table retrieval tasks. Furthermore, Kweon et al. [140] shared similar findings on the Open-WikiTables dataset. Despite the efforts of applying dense retrievers on tables, a number of existing works rely on sparse methods to realize the retriever. Schlichtkrull et al. [165] leveraged TF-IDF to retrieve tables in the open-domain setting. Aly et al. [159] introduce the FEVEROUS dataset and also use a sparse method DrQA [166] to retrieve both text and tables. Moreover, DCUF [167] utilized DrQA to retrieve and BM25 to rerank the retrieved documents. Chen et al. [146] proposed to use fine-tuned DPR on open-domain table retrieval task, and the answers are heavily dependent on the generated SQL queries. These dependencies can result in subpar retrieval performance for unseen tables and cannot handle the cases when the generated SQL queries fail to execute. In summary, effectively retrieving relevant tables from a large data corpus still remains an open question.

Retrieval Augmented Generators (RAG). RAGs leverage retrievers to fetch information from external knowledge database and augment the text input to the language models [168]. REALM [169] and RETRO [149] pretrains the retrieval augmentation framework in the end-to-end manner so that the language model learns to better utilize the retrieved information, while ATLAS [170] jointly trains the retriever as well as the language model. We point out that these well-known RAG models are specialized in the text reasoning domain and cannot be directly applied to table reasoning tasks without fine-tuning, thus inapplicable to the experimental setup of this work.

5.7 Conclusion

In this work, we study the task of open-domain table reasoning, where complex questions are answered using knowledge and evidence stored in structured tables. The reasoning part is empowered by Large Language Models (LLMs), enabling the OPENTAB to work directly on the target dataset without being specifically fine-tuned. This is a highly desired property for several applications, from both efficiency and privacy perspectives. The proposed method OPENTAB incorporates CODER, ROWSELECTOR, and READER that are effective in generating valid SQL programs, highlighting informative content and absorbing longer context allowing for accurate response. We perform comprehensive empirical studies demonstrating the efficacy of OPENTAB. In both open-domain and closed-domain settings, OPENTAB outperforms baseline methods by a large margin, revealing its advanced capability.

5.8 Reproducibility

To ensure reproducibility of OpenTab, we provide detailed discussion regarding the experimental setup in Section 5.4, including the LLM backbone usage, pretrained checkpoint model name, few-shot sample count, etc. We share the prompt used in the experiments in the Appendix. We discuss the details of datasets and the specific metrics used in Section 5.9.3.

5.9 Appendix

5.9.1 Coder in-context learning prompt (2 shots)

```
Given the table schema and three example rows out of the table, write a SQLite program to
↪  extract the sub-table that contains the information needed to answer the questions.
The SQLite does not need to directly answer the question.
Assume you always have enough information when executing the SQLite.
If you cannot generate the SQLite with high confidence that it is correct, then generate some
↪  SQLite that is less complex but correct.
Try to use fuzzy-match for values if you are not sure about the values.
Generate 3 SQLite programs with respect to the question separated by [SQLSEP], the complexities
↪  of the SQLite programs generated ascend (basic, intermediate, advanced).

CREATE TABLE Fabrice_Santoro(
        row_id int,
        name text,
        _2001 text,
        _2002 text,
        _2003 text,
        _2004 text,
        _2005 text,
        _2006 text,
```

```
        _2007 text,
        _2008 text,
        _2009 text,
        _2010 text,
        career_nsr text,
        career_nwin_loss text)
/*
3 example rows:
SELECT * FROM Fabrice_Santoro LIMIT 3;
row_id        name        _2001        _2002        _2003        _2004        _2005        _200⌋
↪  6        _2007        _2008        _2009        _2010        career_nsr        career_nwin_⌋
↪  loss
0   australian open        2r        1r
↪  3r        2r        1r        qf        3r        2r        3r        1r        0 / 18
↪  2218
1        french open        4r        2r        2r        3r        1r        1r        1r⌋
↪          2r        1r        a        0 /
↪  20        1720
2        wimbledon        3r        2r        2r        2r        2r        2r        2r⌋
↪          1r        2r        a        0 /
↪  14        1114
*/
Q: did he win more at the australian open or indian wells?
SQLite:
SELECT name, career_nwin_loss FROM Fabrice_Santoro; [SQLSEP]
SELECT name, career_nwin_loss FROM Fabrice_Santoro WHERE name LIKE "%australian open%" OR name
↪  LIKE "%indian wells%"; [SQLSEP]
WITH Wins AS (
    SELECT
    name,
    CAST(SUBSTR(career_nwin_loss, 1, INSTR(career_nwin_loss, '') - 1) AS INT) AS wins,
    CAST(SUBSTR(career_nwin_loss, INSTR(career_nwin_loss, '') + 1) AS INT) AS losses
    FROM Fabrice_Santoro
    WHERE name LIKE "%australian open%" OR name LIKE "%indian wells%"
```

```
)
SELECT name, SUM(wins) as total_wins, SUM(losses) as total_losses FROM Wins GROUP BY name;



CREATE TABLE Playa_de_Oro_International_Airport(
        row_id int,
        rank text,
        city text,
        passengers text,
        ranking text,
        airline text)
/*
3 example rows:
SELECT * FROM Playa_de_Oro_International_Airport LIMIT 3;
row_id    rank    city    passengers    ranking    airline
0    1    united states, los angeles    14,749    nan    alaska airlines
1    2    united states, houston    5,465    nan    united express
2    3    canada, calgary    3,761    nan    air transat, westjet
*/
Q: how many more passengers flew to los angeles than to saskatoon from manzanillo airport in
↪  2013?
SQLite:
SELECT city, passengers FROM Playa_de_Oro_International_Airport; [SQLSEP]
SELECT city, passengers FROM Playa_de_Oro_International_Airport WHERE city LIKE "%los angeles%"
↪  OR city LIKE "%saskatoon%"; [SQLSEP]
WITH PassengerCounts AS (
    SELECT
    city,
    CAST(REPLACE(passengers, ',', '') AS INT) AS passenger_count
    FROM Playa_de_Oro_International_Airport
    WHERE city LIKE "%los angeles%" OR city LIKE "%saskatoon%"
)
SELECT
SUM(CASE WHEN city LIKE "%los angeles%" THEN passenger_count ELSE 0 END) -
```

```
SUM(CASE WHEN city LIKE "%saskatoon%" THEN passenger_count ELSE 0 END) AS passenger_difference
FROM PassengerCounts;
```

5.9.2 READER in-context learning prompt (2 shots)

```
Given the execution result attained by running SQLite, extract the final answer of the question
↪  from the table.
Assume you can always find the answer from the table, so you must give an answer that makes
↪  sense to the question based on the given table.
If the answer contains multiple items, separate them by [SEP].

CREATE TABLE Fabrice_Santoro(
        row_id int,
        name text,
        _2001 text,
        _2002 text,
        _2003 text,
        _2004 text,
        _2005 text,
        _2006 text,
        _2007 text,
        _2008 text,
        _2009 text,
        _2010 text,
        career_nsr text,
        career_nwin_loss text)
/*
3 example rows:
SELECT * FROM Fabrice_Santoro LIMIT 3;
```

```
row_id        name        _2001        _2002        _2003        _2004        _2005        _200⌋
↪  6        _2007        _2008        _2009        _2010        career_nsr        career_nwin_⌋
↪  loss
0        australian open        2r        1r
↪  3r        2r        1r        qf        3r        2r        3r        1r        0 /
↪  18        2218
1        french open        4r        2r        2r        3r        1r        1r        1r⌋
↪          2r        1r        a        0 /
↪  20        1720
2        wimbledon        3r        2r        2r        2r        2r        2r        2r⌋
↪          1r        2r        a        0 /
↪  14        1114
*/
Q: did he win more at the australian open or indian wells?
SQLite:
SELECT
    name,
    CAST(SUBSTR(career_nwin_loss, 1, INSTR(career_nwin_loss, '-') - 1) AS INT) AS total_wins
FROM
    Fabrice_Santoro
WHERE
    (name LIKE '%australian open%') OR
    (name LIKE '%indian wells%')
Execution Result:
name    total_wins
australian open 22
indian wells    23
A:
indian wells



CREATE TABLE Playa_de_Oro_International_Airport(
        row_id int,
        rank text,
```

```
        city text,
        passengers text,
        ranking text,
        airline text)
/*
3 example rows:
SELECT * FROM Playa_de_Oro_International_Airport LIMIT 3;
row_id    rank    city    passengers    ranking    airline
0    1    united states, los angeles    14,749    nan    alaska airlines
1    2    united states, houston    5,465    nan    united express
2    3    canada, calgary    3,761    nan    air transat, westjet
*/
Q: how many more passengers flew to los angeles than to saskatoon from manzanillo airport in
↪  2013?
SQLite:
SELECT
    city,
    REPLACE(passengers, ',', '') AS passenger_count
FROM
    Playa_de_Oro_International_Airport
WHERE
    (city LIKE '%los angeles%') OR
    (city LIKE '%saskatoon%')
Execution Result:
city    passenger_count
united states, los angeles  14749
mexico, saskatoon  10000
A:
4749
```

5.9.3 Datasets

Open-WikiTables. We directly leverage the Open-WikiTables dataset provided by [140] in our experiments. Note that this dataset has limited amount of cases whose final ground truth answer is empty, which will not be used in our experiments. Our evaluations happen on the validation set of the dataset. Due to the budget limitation, the experiments are carried out on 2,000 random samples out of the validation set with a fixed random seed 42 to make sure all the models and methods are evaluated on the same subset fairly. Note that for this dataset each input question will only has one gold table as evidence. Table corpus size is 24,680. For the evaluation metric, we adopt the *semantic-match* evaluator as used in BINDER [141], which is a more reasonable metric to evaluate the table QA task.

FEVEROUS. FEVEROUS [159] dataset is a fact verification dataset consisting of evidences in the format of both natural language text and structured tables. To adapt the dataset into the open-domain table reasoning setting, we filter 323 claims out of the validation set whose verification is only based on table information. Also because the tables are noisy web tables where a lot of samples incorporate undesired properties such as empty sells, multi-header, cell count mismatch, etc. These tables cannot naturally be used in the SQL database interface so we need to further remove claims that need to be verified on such tables. The table corpus of the FEVEROUS dataset used here is of 26,177 different tables. Note that for the processed subset, each claim will only have one gold table as evidence. The prediction space is limited to “refutes” and “supports”. For the evaluation metric, inspired by the FEVEROUS score metric in Aly et al. [159], which requires predictions to be made on the correct evidences, we further invent our metric as to make the right prediction while locating the correct evidence table in the same time. Such scenario will be

Table 5.7: Applicability of OPENTAB and baselines.

Method	Open-domain	Large table	w/o Fine-tuning
FEVEROUS-base [159]	✓		
DCUF [167]	✓		
OpenWT-base [140]	✓		
BINDER [141]			✓
DATER [142]			✓
OPENTAB (proposed)	✓	✓	✓

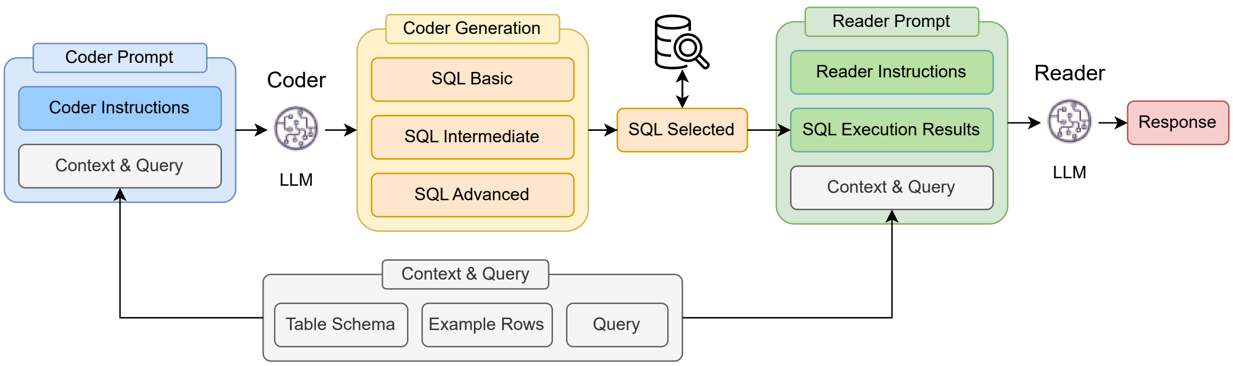


Figure 5.5: Prompt and generation structures of both CODER and READER.

regarded as a successful prediction. As long as the model does not make the right decision, or makes decision based on the wrong table, it will be considered a failure case.

5.9.4 Sanity check on text-to-SQL dataset

STC is proposed in the context of end-to-end table QA task, in which SQL programs work as an efficient and scalable interface to query the database and extract relevant information for READER to make inference on. SQL generation is an intermediate tool in the QA pipeline, instead of outcome. While on text-to-SQL benchmarks, ground truth is SQL program itself, which is different from our motivation. While text-to-SQL task is not the focus of this work, we carried out complementary experiments on the "flight_4" database of the Spider dataset to verify the effectiveness of our CODER module with the STC prompting strategy. Below we list the results in the order of `question`, `ground truth`, and `SQL generated by CODER`. We also show

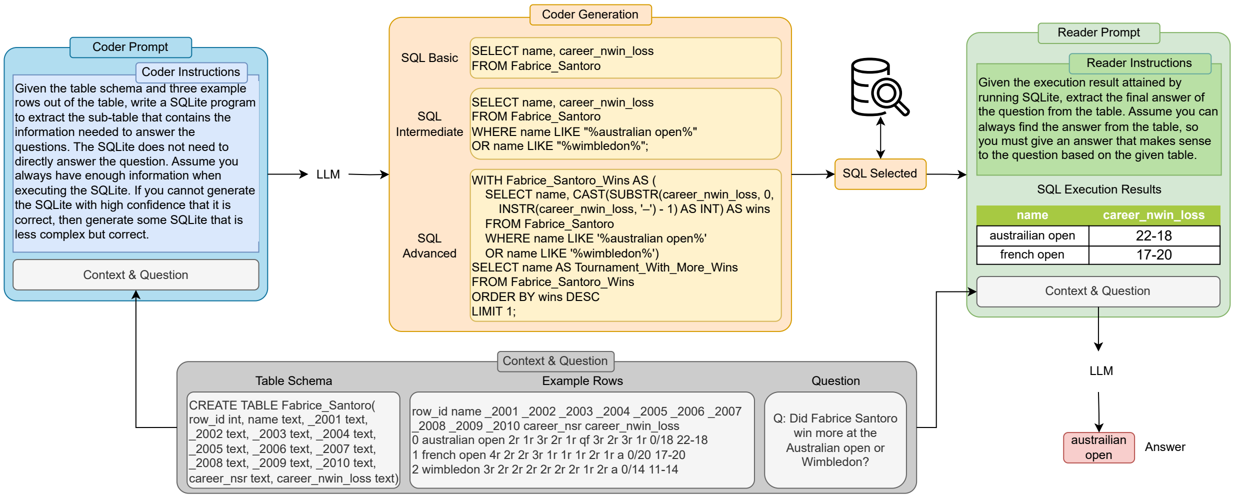


Figure 5.6: Concrete prompt and generation structures of both CODER and READER.

the stats of the large-scale database in the table.

```
How many routes does American Airlines operate?
SELECT count(*) FROM airlines AS T1 JOIN routes AS T2 ON T1.alid = T2.alid WHERE T1.name =
↪ 'American Airlines'
SELECT COUNT(*) FROM routes r INNER JOIN airlines a ON r.alid = a.alid WHERE a.name = 'American
↪ Airlines';
```

```
How many routes end in a Canadian airport?
SELECT count(*) FROM airports AS T1 JOIN routes AS T2 ON T1.apid = T2.dst_apid WHERE country =
↪ 'Canada'
SELECT COUNT(*) FROM routes WHERE dst_apid IN (SELECT apid FROM airports WHERE country =
↪ 'canada');
```

```
What are the names of all airports in Cuba or Argentina?
SELECT name FROM airports WHERE country  =  'Cuba' OR country  =  'Argentina'
SELECT name FROM airports WHERE country LIKE "%cuba%" OR country LIKE "%argentina%";
```

```
What are the countries of all airlines whose names start with Orbit?
```

```
SELECT country FROM airlines WHERE name LIKE 'Orbit%'

SELECT DISTINCT country FROM airlines WHERE name LIKE "Orbit%";
```

In how many cities are there airports in the country of Greenland?

```
SELECT count(DISTINCT city) FROM airports WHERE country = 'Greenland'

SELECT COUNT(DISTINCT city) as airport_cities FROM airports WHERE country LIKE "%greenland%";
```

Bibliography

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[2] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *Advances in Neural Information Processing Systems*, 34:16410–16422, 2021.

[4] Mucong Ding, Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Micah Goldblum, David Wipf, Furong Huang, and Tom Goldstein. A closer look at distribution shifts and out-of-distribution generalization on graphs. 2021.

[5] Renkun Ni, Micah Goldblum, Amr Sharaf, Kezhi Kong, and Tom Goldstein. Data augmentation for meta-learning. In *International Conference on Machine Learning*, pages 8152–8161. PMLR, 2021.

[6] Hao-Zhe Feng, Kezhi Kong, Minghao Chen, Tianye Zhang, Minfeng Zhu, and Wei Chen. Shot-vae: semi-supervised deep generative models with label-aware elbo approximations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7413–7421, 2021.

[7] Yantao Shen, Hongsheng Li, Shuai Yi, Dapeng Chen, and Xiaogang Wang. Person re-identification with deep similarity-guided graph neural network. In *Proceedings of the European conference on computer vision (ECCV)*, pages 486–504, 2018.

[8] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

[9] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119, 2018.

[10] Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2594–2604, 2019.

[11] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[12] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[15] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[16] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[18] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[19] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33, 2020.

[20] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 207–217, 2020.

[21] Jonathan Godwin, Michael Schaarschmidt, Alexander L Gaunt, Alvaro Sanchez-Gonzalez, Yulia Rubanova, Petar Veličković, James Kirkpatrick, and Peter Battaglia. Simple GNN regularisation for 3d molecular property prediction and beyond. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=1wVvweK3oIb`.

[22] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Robust optimization as data augmentation for large-scale graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 60–69, 2022.

[23] Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. Vq-gnn: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C Bayan Bruss, and Tom Goldstein. Goat: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, pages 17375–17390. PMLR, 2023.

[25] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. On the reliability of watermarks for large language models. *arXiv preprint arXiv:2306.04634*, 2023.

[26] Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Opentab: Advancing large language models as open-domain table reasoners. *arXiv preprint arXiv:2402.14361*, 2024.

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

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

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

[30] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

[31] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[32] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2018.

[33] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[34] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *NeurIPS*, 2018.

[35] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

[37] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[38] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, pages 10764–10799. PMLR, 2023.

[39] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[40] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[41] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[42] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[43] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv preprint arXiv:1910.08051*, 2019.

[44] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *Advances in neural information processing systems*, pages 5334–5344, 2018.

[45] Manli Shu, Zuxuan Wu, Micah Goldblum, and Tom Goldstein. Prepare for the worst: Generalizing across domain shifts with adversarial batch normalization. *arXiv e-prints*, pages arXiv–2009, 2020.

[46] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 819–828, 2020.

[47] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*, 2016.

[48] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Thomas Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

[49] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

[50] Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-scale adversarial training for vision-and-language representation learning. *arXiv preprint arXiv:2006.06195*, 2020.

[51] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3358–3369, 2019.

[52] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[53] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[54] Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

[55] Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[56] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[57] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[58] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[59] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[60] Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

[61] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

[62] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

[63] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[64] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *arXiv preprint arXiv:1911.07323*, 2019.

[65] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *NeurIPS*, 33, 2020.

[66] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 2018.

[67] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[68] OmriPuny HeliBen-Hamu YaronLipman. Global attention improves graph networks generalization. *arXiv preprint arXiv:2006.07846*, 2020.

[69] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *NeurIPS*, 33, 2020.

[70] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[71] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

[72] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, pages 1993–2001, 2016.

[73] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[74] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

[75] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *ICLR*, 2020.

[76] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *ICLR*, 2020.

[77] Muhammet Balcilar, Renton Guillaume, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*, 2021.

[78] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[79] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[80] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[81] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[82] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[83] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *ICLR*, 2015.

[84] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*, page 29, 2004.

[85] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[86] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020.

[87] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[88] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000*, 2020.

[89] Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. Sgquant: Squeezing the last bit on graph neural networks with specialized quantization. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1044–1052. IEEE, 2020.

[90] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

[91] Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. Biva: A very deep hierarchy of latent variables for generative modeling. *arXiv preprint arXiv:1902.02102*, 2019.

[92] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020.

[93] Alexei Baevski, Steffen Schneider, and Michael Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. *arXiv preprint arXiv:1910.05453*, 2019.

[94] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*, 2020.

[95] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[96] Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is attention better than matrix decomposition. In *International Conference on Learning Representations*, 2021.

[97] Hanwei Wu and Markus Flierl. Learning product codebooks using vector-quantized autoencoders for image retrieval. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[98] David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*, 2018.

[99] George Dasoulas, Kevin Scaman, and Aladin Virmaux. Lipschitz normalization for self-attention layers with application to graph neural networks. *arXiv preprint arXiv:2103.04886*, 2021.

[100] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. *arXiv preprint arXiv:2106.05609*, 2021.

[101] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[102] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[103] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[104] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34, 2021.

[105] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

[106] Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Edge-augmented graph transformers: Global self-attention is enough for graphs. *arXiv preprint arXiv:2108.03348*, 2021.

[107] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[108] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34, 2021.

[109] Lukasz Maziarka, Dawid Majchrowski, Tomasz Danel, Piotr Gaiński, Jacek Tabor, Igor Podolak, Pawel Morkisz, and Stanislaw Jastrzebski. Relative molecule self-attention transformer. *arXiv preprint arXiv:2110.05841*, 2021.

[110] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[111] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

[112] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *Advances in Neural Information Processing Systems*, 34, 2021.

[113] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[114] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. *Advances in Neural Information Processing Systems*, 34, 2021.

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

[116] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.

[117] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[118] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[119] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[120] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[121] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

[122] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

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

[124] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[125] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[126] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/xu18c.html`.

[127] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

[128] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

[129] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

[130] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. *arXiv preprint arXiv:2111.00064*, 2021.

[131] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[132] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR, 2020.

[133] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention for transformer models. In *International Conference on Machine Learning*, pages 10183–10192. PMLR, 2021.

[134] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[135] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

[136] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics (ACL)*, pages 1049–1065, 2023.

[137] Xi Ye and Greg Durrett. The unreliability of explanations in few-shot prompting for textual reasoning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 30378–30392, 2022.

[138] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. Tutorial proposal: Retrieval-based language models and applications. In *The 61st Annual Meeting of the Association for Computational Linguistics (ACL): Tutorial Abstracts*, page 41, 2023.

[139] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. Open domain question answering over tables via dense retrieval. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 512–519, 2021.

[140] Sunjun Kweon, Yeonsu Kwon, Seonhee Cho, Yohan Jo, and Edward Choi. Open-WikiTable : Dataset for open domain question answering with complex reasoning over table. In *Findings of the Association for Computational Linguistics (ACL)*, pages 8285–8297, 2023.

[141] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[142] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 174184, 2023.

[143] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

[144] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. Open question answering over tables and text. In *The Ninth International Conference on Learning Representations (ICLR)*, 2021.

[145] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

[146] Siqin Chen, Yubo Liu, Jie Wu, and Mengshu Hou. Retrieval augmented via execution guidance in open-domain table qa. In *Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI)*, pages 1–6, 2022.

[147] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4320–4333, 2020.

[148] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[149] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162, pages 2206–2240. PMLR, 2022.

[150] Wenhu Chen. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics (EACL)*, page 11201130, 2023.

[151] Zhiruo Wang, Zhengbao Jiang, Eric Nyberg, and Graham Neubig. Table retrieval may not necessitate table-specific model design. In *Proceedings of the Workshop on Structured and Unstructured Knowledge Integration (SUKI)*, pages 36–46, 2022.

[152] Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qifei Wu, Yuchen Ding, Hua Wu, Haifeng Wang, and Ji-Rong Wen. A thorough examination on zero-shot dense retrieval. *arXiv preprint arXiv:2204.12755*, 2022.

[153] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[154] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[155] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability, 2023.

[156] Ruoxi Sun, Sercan O. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. Sql-palm: Improved large language model adaptation for text-to-sql, 2023.

[157] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[158] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1470–1480, 2015.

[159] Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. Feverous: Fact extraction and verification over unstructured and structured information. In *Proceedings of the Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, volume 1, 2021.

[160] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 602–631, 2022.

[161] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. TAPEX: Table pre-training via learning a neural SQL executor. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022.

[162] Fan Zhou, Mengkang Hu, Haoyu Dong, Zhoujun Cheng, Shi Han, and Dongmei Zhang. Tacube: Pre-computing data cubes for answering numerical-reasoning questions over tabular data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2278–2291, 2022.

[163] Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 932–942, 2022.

[164] Hangfeng He, Hongming Zhang, and Dan Roth. Rethinking with retrieval: Faithful large language model inference. *arXiv preprint arXiv:2301.00303*, 2022.

[165] Michael Sejr Schlichtkrull, Vladimir Karpukhin, Barlas Oguz, Mike Lewis, Wen-tau Yih, and Sebastian Riedel. Joint verification and reranking for open fact checking over tables. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 6787–6799, 2021.

[166] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1870–1879, 2017.

[167] Nan Hu, Zirui Wu, Yuxuan Lai, Xiao Liu, and Yansong Feng. Dual-channel evidence fusion for fact verification over texts and tables. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 5232–5242, 2022.

[168] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[169] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.

[170] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.