



SAARLAND UNIVERSITY
Department of Language Science and Technology

SEMINAR
Machine Learning for Natural Language Processing

A Generalization of Transformer Networks to Graphs

Author:
Jaykumar BHAGIYA
7055903

Supervisor:
Dr. Dietrich KLAOW

April 13, 2025

Abstract

This technical report explores a novel adaptation of the transformer architecture, originally developed for natural language processing, to effectively process arbitrary graph structures. Unlike the standard transformer, which treats input tokens as a fully connected sequence and excels in linear data arrangements, it struggles to capture the inherent connectivity of graphs due to its lack of structural inductive bias and limited node feature integration. To address this, the proposed Graph Transformer architecture modifies four key components of the standard model. First, the attention mechanism is enhanced to incorporate function of neighborhood connectivity for each node in the graph. Second, positional encoding is re-defined using graph-specific Laplacian eigenvectors, improving the representation of structural properties and generalizing the sinusoidal encoding typical in sequential data. Third, layer normalization is replaced with batch normalization to accelerate training and enhance validation performance. Finally, the architecture is extended to include edge feature representations, enabling it to model critical relationships in applications such as social networks (interaction strengths) or transportation systems (connection weights).

The Graph Transformer is evaluated through numerical experiments on established graph benchmarks, showcasing its effectiveness in handling diverse graph-based tasks. This work bridges the divide between traditional transformers, optimized for sequential data, and graph neural networks, designed for arbitrary topologies. The resulting architecture offers a simple and generic framework, suitable for applications requiring transformer and graph.

Contents

Abstract	i
1 Introduction	1
2 Background	2
2.1 Transformer	2
2.1.1 Self-Attention	2
2.1.2 Positional Encoding	3
2.2 Graph Neural Networks (GNNs)	4
2.3 Related Work	4
3 Key Design Aspects	6
3.1 Graph Sparsity	6
3.2 Positional Encodings (PEs)	7
3.2.1 Intuition Behind Using Laplacian Eigenvectors	7
4 Proposed Architecture	9
4.1 Input	10
4.2 Graph Transformer Layer	10
4.3 Graph Transformer Layer with edge features	10
4.4 Task based MLP Layers	11
5 Numerical Experiments	12
5.1 Addressing Questions	12
5.2 Limitations	14
6 Conclusion	15
A Transformer	16
B Task based MLP layer equations	17
Bibliography	18

Chapter 1

Introduction

Transformers (Vaswani et al., 2017) have become the dominant neural network architecture in natural language processing (NLP), excelling at language modeling and a wide range of language-related tasks across multiple languages. They rely on a self-attention mechanism (Bahdanau, Cho, and Bengio, 2014), which enables each word in a sentence to attend to all other words, generating context-rich feature representations. This ability to capture long-range dependencies has made transformer-based architectures a foundational building block in modern deep learning. Models leveraging this self-attention mechanism, such as GPT (Radford et al., 2018) and BERT (Devlin et al., 2019), are widely used in NLP applications, including text generation, sentiment analysis, and machine translation.

On the other hand, graph neural networks (GNNs) have emerged as the leading deep learning approach for graph-structured data. GNNs are particularly effective for tasks where relationships between data points are naturally represented as graphs. Using the message-passing paradigm (Gilmer et al., 2017), GNNs iteratively update node representations by aggregating information from neighboring nodes. This iterative process allows the network to capture both local and global structural patterns, leading to effective feature abstraction. With their expressive representations, GNNs have achieved success in various tasks, including network analysis prediction in social science (Monti et al., 2019), particle interaction modeling in physics (Cranmer et al., 2019, Sanchez-Gonzalez et al., 2020), and applications in knowledge graphs (Schlichtkrull et al., 2018, Chami et al., 2020).

The primary distinction between natural language processing (NLP) and graph neural networks (GNNs) lies in the nature of the data each field addresses. In NLP, datasets typically follow a sequential structure, such as sequences of words, sentences, or paragraphs, where the order of elements carries significant meaning. Conversely, in graph-related domains, data is represented as arbitrary, non-sequential structures, where nodes and edges form complex networks that vary widely in configuration across different graphs. For instance, a social network graph might connect users through friendships, while a molecular graph links atoms via chemical bonds, each with unique topologies. This fundamental difference in data representation shapes the approaches used in these fields.

Inspired by the remarkable success of transformers in NLP, researchers have explored ways to extend their capabilities beyond sequential data to accommodate datasets with arbitrary graph structures. Transformers, with their self-attention mechanisms, excel at modeling long-range dependencies in sequences, but adapting them to graphs requires rethinking how attention operates over non-linear relationships. This generalization empowers transformers to process graph datasets by treating nodes as elements that attend to one another based on their connectivity, thus broadening their applicability across diverse domains.

Chapter 2

Background

2.1 Transformer

Our understanding of a word is shaped by the context provided by other words in a sentence, and transformers replicate this by assigning attention weights to all words in a sequence, emphasizing those most relevant for understanding the context. Two key components—Self-Attention and Positional Encoding—are briefly explained here, adapted from the original transformer model and later updated in the proposed Graph Transformer. The complete Transformer architecture is shown in Figure A.1, in Appendix A.

2.1.1 Self-Attention

The self-attention mechanism in transformer models captures relationships between elements in a sequence, such as words in NLP. It involves two primary steps: first, a scoring function computes an alignment score between a pair of elements (Query and Key), which is then normalized using the Softmax function to produce a probability distribution. This distribution is used to calculate a weighted sum over another component, the Value, resulting in the attention output. The attention mechanism can be formally expressed as (here d^k size Query matrix):

$$Attention(Q, K, V) = softmax \left(\frac{QK^T}{\sqrt{d^k}} \right) \cdot V$$

In multi-head attention, the mechanism is extended to allow the model to focus on various aspects of the sequence. This is done by splitting the Query, Key, and Value into multiple subspaces, processing them through separate attention heads, and concatenating the results. This approach enriches the model's ability to capture diverse contextual dependencies. The process is mathematically represented by the following equations:

$$w_{ij}^{k,l} = softmax_j \left(\frac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right)$$

$$\hat{h}_i^{l+1} = O_h^l \parallel \left(\sum_{k=1}^H \sum_{j \in S} w_{ij}^{k,l} V^{k,l} h_j^l \right)$$

Here, l is the current layer, H denotes attention heads, and $\parallel$ represents concatenation of attention head outputs. h_i^l and h_j^l are the i th and j th words in the input sequence, with $w_{ij}^{k,l}$

being the attention score between them. The matrices $Q^{k,l}$, $K^{k,l}$, $V^{k,l}$, and O_h^l are trainable parameters.

2.1.2 Positional Encoding

Positional Encoding (PE) is introduced to address the transformer's limitation of not considering the order of elements in a sequence. Without it, the model treats the input as an unordered set, which is detrimental to understanding sequential data like natural language. To overcome this, positional encodings are added, which provide information about the position of each word in the sequence. These encodings are generated using sine and cosine functions of different frequencies, calculated using the following formulas:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

Figure 2.1 illustrates the positional encodings with varying frequencies, where $d_{\text{model}} = 4$ and pos represents the word position. Adding these unique position-based vectors to word embeddings enables the model to capture the relative positioning and relationships between words in a sequence.

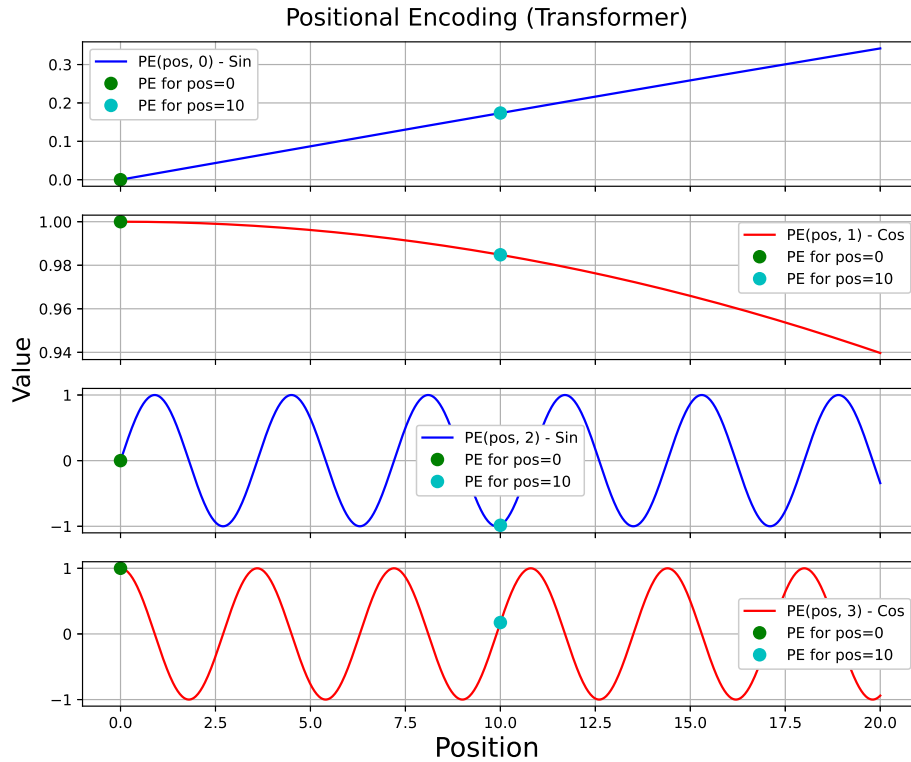


FIGURE 2.1: Positional Encodings Using Trigonometric Functions

2.2 Graph Neural Networks (GNNs)

Graphs provide a natural representation for complex data structures involving entities and their interactions, where entities are modeled as nodes and relationships as edges. This flexibility allows graphs to be applied to a wide range of tasks. However, integrating graph structures into neural networks presents challenges, as most traditional models are designed for fixed-size arrays or grid-like data, such as images or sequences, rather than irregular structures like graphs.

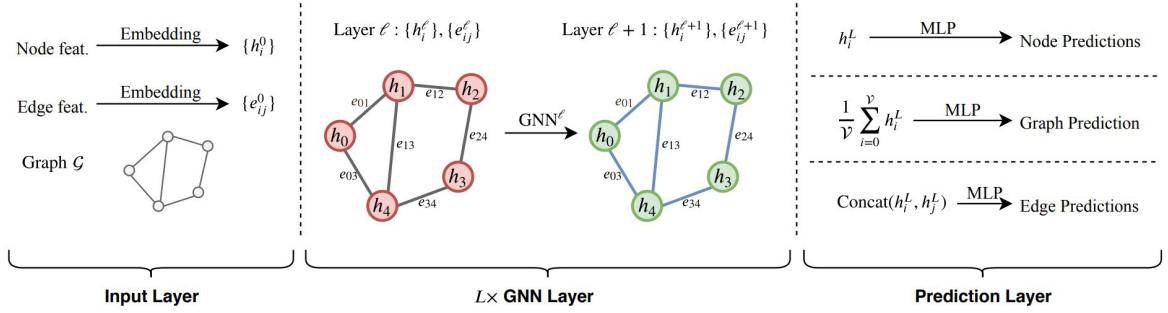


FIGURE 2.2: End-to-end schematic diagram of a L layer GNN model, in which every layer builds abstract representations of nodes and edges using the message-passing framework, i.e. nodes exchange messages with their first-hop neighbors at every layer

In graph-based tasks, three key types of information are utilized: node features, edge attributes, and the graph's overall structure. Node features can be represented by a feature matrix, where each row corresponds to a node and its feature vector. For edges, an adjacency matrix is commonly used to define connectivity between nodes, though this method has limitations. These numerical graph representations serve as input to Graph Neural Networks (GNNs), which iteratively refine node and edge features through message passing (Gilmer et al., 2017) while maintaining the graph's structure.

GNNs operate by preserving graph topology during learning, updating node and edge embeddings while keeping the original graph structure intact. Stacking multiple GNN layers helps to enhance learning. After the GNN layers, each component of the graph (node, edge, and global attributes) is processed through a separate Multi-Layer Perceptron (MLP) to produce updated representations. Pooling operations within GNN layers can also help improve the model's sensitivity to graph structure by enabling nodes to aggregate information from their neighbors, refining their embeddings based on the graph's topology. An overview of this architecture is depicted in Figure 2.2, as shown in the paper (Dwivedi et al., 2023).

2.3 Related Work

Both Natural Language Processing (NLP) and Graph Neural Networks (GNNs) are active areas of research, and numerous efforts have been made to extend transformer architectures to graph-structured data. These efforts include exploring the benefits of localized attention mechanisms, identifying effective positional embedding strategies, and developing

transformer models tailored for specific scenarios such as temporal graphs, heterogeneous graphs, and generative models.

The architecture proposed by Li et al. (Li et al., 2019) adopts global attention—where each node attends to all other nodes in the graph—rather than limiting attention to local neighborhoods. The motivation behind this approach is to capture comprehensive, global structural information. However, this design choice introduces potential challenges, such as high computational complexity on large graphs and the loss of sparsity—a property that often provides a beneficial inductive bias in graph-based learning. To mitigate these issues, alternative strategies have been proposed for incorporating global context without sacrificing local structure or sparsity. For instance, graph-specific positional features (Zhang et al., 2020) or Laplacian positional encodings using eigenvectors (Belkin and Niyogi, 2003, Dwivedi et al., 2023) offer effective ways to encode global information while maintaining the graph’s structural integrity.

Graph-BERT (Zhang et al., 2020) presents another notable approach. It employs a pre-training strategy based on a subgraph batching scheme that constructs fixed-size, linkless subgraphs. This design facilitates parallel learning and scalability. Moreover, it integrates a hybrid of positional encoding methods to capture both absolute and relative structural positions of nodes, enhancing the model’s ability to learn from the graph’s topology.

Graph Transformer Networks (GTNs) (Yun et al., 2019) address heterogeneity in graphs by transforming them into meta-path-based representations. Meta-paths are defined as specific sequences of node and edge types, enabling the model to learn semantic relationships through an attention mechanism. In contrast, the Heterogeneous Graph Transformer (HGT) (Hu et al., 2020) is built to handle heterogeneous information networks with an arbitrary number of node and edge types. HGT incorporates relative temporal positional encodings—derived from timestamp differences between central and message-passing nodes—to effectively model the dynamic flow of information in temporal and evolving graphs.

A transformer-based generative approach is proposed by Zhang et al. (Zhang et al., 2020) for modeling temporal graphs, learning to capture evolving patterns directly from dynamic network structures. Additionally, the Universal Graph Transformer framework introduced by Nguyen et al. (Nguyen, Nguyen, and Phung, 2019) targets uniform graphs and employs embedding-based positional encoding to define node positions. However, their results indicate that this strategy performs inconsistently, showing notable improvements only in a few unsupervised learning tasks.

Despite the progress made, several open questions remain. Should attention mechanisms be confined to local neighborhoods, or does global attention consistently lead to better performance? What positional encoding strategies best capture the complexities of graph-structured data? Since positional encodings are crucial for embedding structural information into transformer models, their design remains a pivotal consideration. Moreover, how can challenges related to scalability and pretraining be effectively addressed to ensure the practical deployment of graph transformers?

Chapter 3

Key Design Aspects

Designing an effective graph transformer architecture requires adapting the principles of traditional transformers to the unique challenges of graph-structured data. The goal is to generalize key components of transformers, such as attention mechanisms, beyond sequential inputs to arbitrary graph topologies. A natural starting point is the attention mechanism, which in standard transformers is global, allowing each token to attend to every other token. However, in graphs, where relationships are sparse, global attention can be computationally expensive and less meaningful. To address this, many graph-based models, such as Graph Attention Networks (GATs) (Veličković et al., 2017) and discussed in related work section, limit attention to immediate neighbors, aligning better with graph structures.

While local attention is a fundamental adaptation for graph-based models and has demonstrated practical success, it alone may not suffice for building a fully expressive and generalized graph transformer. The authors argue that other considerations must be integrated into the architecture to capture richer structural information and maintain scalability. Two critical components are **graph sparsity** and **positional encodings**, which help the model better represent graph structures while maintaining computational efficiency.

3.1 Graph Sparsity

In natural language processing (NLP), transformers use full attention, justified by two main factors. First, sentences lack inherent sparse structure, meaning there is no predefined connectivity between words, and relationships depend heavily on context. For instance, in the sentence "Welcome to the Seminar," the relationship between "welcome" and "the" is unclear without context. Second, NLP sentences are typically short, containing only tens or hundreds of words, making full attention computationally feasible.

In contrast, graphs inherently possess rich structural connectivity, which provides valuable domain-specific information. Unlike sentences, graphs have predefined relationships between nodes, such as chemical bonds in molecular graphs or friendships in social networks. Ignoring this structure would lead to a loss of critical inductive biases needed for effective learning. Additionally, applying full attention to graphs presents two key challenges: the complexity of graph structure and scalability issues, as graphs can have thousands to millions of nodes, making full attention computationally impractical.

To address these challenges, sparse attention, which focuses on a node's local neighborhood, offers an effective solution. This approach aligns with the principles of Graph Neural Networks (GNNs), where feature aggregation relies on local connectivity. By leveraging

graph structure, sparse attention serves as a powerful inductive bias, enhancing the model's ability to generalize better.

3.2 Positional Encodings (PEs)

Positional Encodings serves an additional aspect to mitigate the limitations of local attention to incorporate global context. Unlike sentences, graphs do not have a natural linear sequence. Instead, their structure is defined by their connectivity. Literature (e.g., Srinivasan and Ribeiro, 2019, You, Ying, and Leskovec, 2019, and Dwivedi et al., 2023) has shown that including PEs into graph neural networks (GNNs) significantly improves their performance. PEs help address some fundamental limitations of GNNs, such as their inability to: Distinguish between non-isomorphic graphs and Detect cycles or other structural patterns within graphs.

Given the importance of PEs in graph-based models, the authors of this work extend this concept when designing the Graph Transformer. Specifically, they use Laplacian eigenvectors as PEs from Spectral graph theory. This approach builds upon their previous work (Dwivedi et al., 2023), where Laplacian eigenvectors were shown to be highly effective in encoding graph structure for various graph learning tasks.

3.2.1 Intuition Behind Using Laplacian Eigenvectors

The Laplacian matrix (L) encodes the graph's structure and is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix. The matrix is positive semi-definite, meaning all its eigenvalues are non-negative ($\lambda \geq 0$).

The eigenvectors of L satisfy the equation $Lu = \lambda u$, where u is an eigenvector and λ is the corresponding eigenvalue. These eigenvalues provide insights into the graph's structural properties.

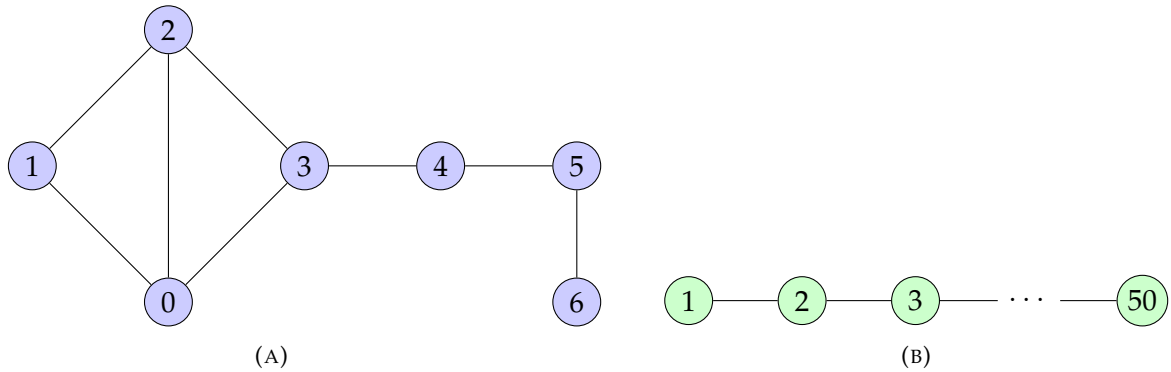


FIGURE 3.1: Here, (A) is Graph used to demonstrate the The Fiedler vector, while (B) is Sequential Graph used to demonstrate generalization of the sinusoidal PEs

The Fiedler vector is the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix. It is particularly useful for understanding the graph's structure, as it captures information about the connectivity and division of the graph into distinct clusters. The values of the Fiedler vector (Second plot in figure 3.2) provide a way to partition the

graph: nodes with negative values (red dots) in the Fiedler vector correspond to one cluster (the left side of the graph A in figure 3.1), while nodes with positive values (green dots) correspond to another cluster (the right side of the graph A in figure 3.1). This reflects the graph's structure, with the eigenvector revealing how well the graph is connected and how nodes can be grouped based on their connectivity.

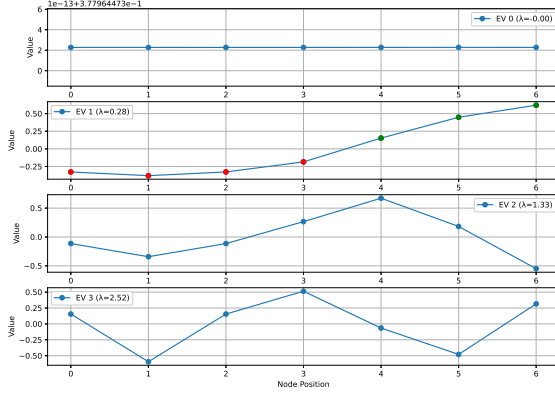


FIGURE 3.2: Laplacian Eigenvectors as Positional Encodings in Graphs

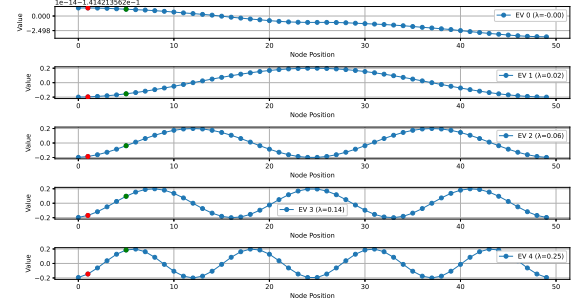


FIGURE 3.3: Positional Encodings Using Laplacian Eigenvectors for Sequential Graph

Laplacian eigenvectors represent different frequency components of the graph:

- **Global Relationships:** The eigenvectors corresponding to smaller eigenvalues vary slowly, capturing broad, global structures of the graph. These provide a smooth positional encoding that reflects global relationships between distant nodes.
- **Local Relationships:** Eigenvectors with larger eigenvalues vary more rapidly, capturing fine, local details of the graph structure.

Further Laplacian eigenvectors serve as a natural generalization of the sinusoidal positional encodings used in NLP transformers. Figure 3.3 plots PEs of graph B from figure 3.1. PEs in figure 3.3 closely resembles PEs used in NLP (see figure 2.1). To understand further, consider the following:

- In NLP transformers, the sinusoidal positional encodings can be interpreted as eigenvectors of the line graph of the sentence.
- In graph transformers, this idea is extended by using the eigenvectors of the graph's Laplacian matrix.

This connection demonstrates a seamless extension of the positional encoding concept from NLP to graph learning. By encoding the structural information of the graph, Laplacian eigenvectors enable the Graph Transformer to effectively capture graph-specific properties.

Chapter 4

Proposed Architecture

This section introduces the architecture of the Graph Transformer Layer, including a variant that incorporates edge features. As shown in Figure 4.1, the first model configuration is tailored for graphs without explicit edge information, operating solely on node features. The second version extends this by integrating a dedicated edge feature pipeline, enabling the model to include edge information and maintain its abstract representations across all layers.

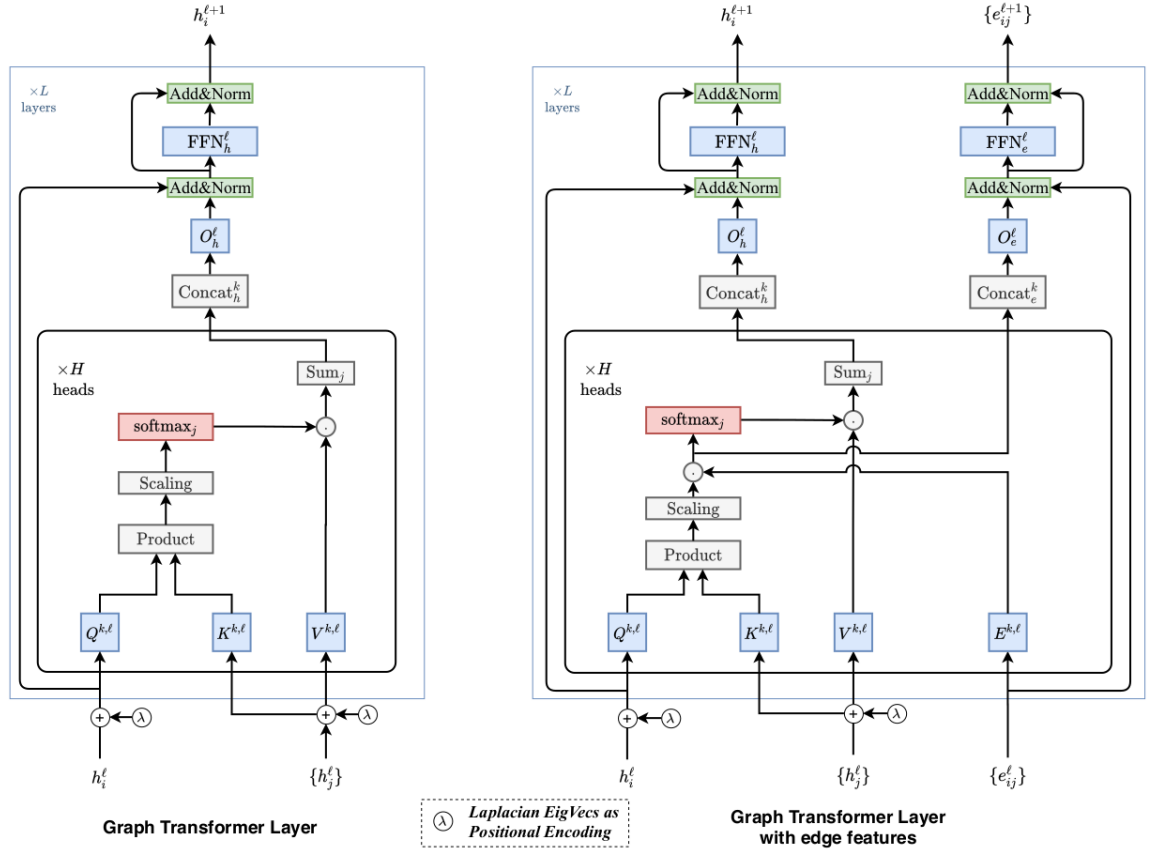


FIGURE 4.1: Block diagram of the Graph Transformer architecture **Left:** A variant of the Graph Transformer that operates solely on node features to calculate attention scores. **Right:** An extended version of the model that includes a dedicated edge feature pipeline

4.1 Input

To prepare the input, initial node and edge features are first linearly projected into a shared hidden dimensional space. Given a graph $\mathcal{G}$ with node features $\alpha_i \in \mathbb{R}^{d_n \times 1}$ for each node i , and edge features $\beta_{ij} \in \mathbb{R}^{d_e \times 1}$ for each edge between node i and node j , the transformation is performed using trainable projection matrices and bias terms as follows:

$$\hat{h}_i^0 = A^0 \alpha_i + a^0; \quad e_{ij}^0 = B^0 \beta_{ij} + b^0$$

where A^0, B^0 are projection matrices and a^0, b^0 are bias vectors. Additionally, node features are augmented with positional encodings derived from the Laplacian eigenvectors, which are embedded using a separate projection and followed by summation with node features:

$$\lambda_i^0 = C^0 \lambda_i + c^0; \quad h_i^0 = \hat{h}_i^0 + \lambda_i^0$$

It is important to note that these positional encodings are applied only at the input layer and are not propagated through subsequent layers of the network.

4.2 Graph Transformer Layer

The Graph Transformer architecture follows a structure similar to the original transformer model, as illustrated in Figure 4.1 (Left). The formulation below outlines the process for updating node representations at a given layer ℓ .

$$\hat{h}_i^{\ell+1} = O_h^\ell \left\| \sum_{k=1}^H \left(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^\ell \right) \right.$$

$$\text{where } w_{ij}^{k,\ell} = \text{softmax}_j \left(\frac{Q^{k,\ell} h_i^\ell \cdot K^{k,\ell} h_j^\ell}{\sqrt{d_k}} \right),$$

In this formulation, $Q^{k,\ell}, K^{k,\ell}, V^{k,\ell} \in \mathbb{R}^{d_k \times d}$ represent the learnable projection matrices corresponding to queries, keys, and values for each of the H attention heads, while $O_h^\ell \in \mathbb{R}^{d \times d}$ is the output projection matrix. The resulting attention output $\hat{h}_i^{\ell+1}$ is subsequently passed through a feed-forward network (FFN), with residual connections and layer normalization applied both before and after the FFN block.

4.3 Graph Transformer Layer with edge features

The Graph Transformer extended with edge features is specifically designed to utilize pairwise attribute information often present in graph datasets such as molecules or social networks. As shown in Figure 4.1 (Right), this architecture introduces a mechanism to integrate explicit edge information directly into the attention computation. Instead of relying solely on attention scores derived from query-key interactions, the model refines these intermediate scores by incorporating edge features, thereby enhancing the attention weights with

meaningful pairwise context. Additionally, a separate pipeline is maintained for propagating edge representations across layers, allowing for dynamic updates of edge features throughout the network.

The mathematical formulation for a single layer ℓ is as follows:

$$\begin{aligned}\hat{h}_i^{\ell+1} &= O_h^\ell \parallel_{k=1}^H \left(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^\ell \right), \\ \hat{e}_{ij}^{\ell+1} &= O_e^\ell \parallel_{k=1}^H \left(\hat{w}_{ij}^{k,\ell} \right), \quad \text{with} \\ w_{ij}^{k,\ell} &= \text{softmax}_j \left(\hat{w}_{ij}^{k,\ell} \right), \\ \hat{w}_{ij}^{k,\ell} &= \left(\frac{Q^{k,\ell} h_i^\ell \cdot K^{k,\ell} h_j^\ell}{\sqrt{d_k}} \right) \cdot E^{k,\ell} e_{ij}^\ell\end{aligned}$$

Here, $Q^{k,\ell}, K^{k,\ell}, V^{k,\ell}, E^{k,\ell} \in \mathbb{R}^{d_k \times d}$ are the projection matrices for query, key, value, and edge features, respectively, while $O_h^\ell, O_e^\ell \in \mathbb{R}^{d \times d}$ are the output projections. The resulting node and edge outputs, $\hat{h}_i^{\ell+1}$ and $\hat{e}_{ij}^{\ell+1}$, are further processed through their respective feed-forward layers, each surrounded by residual connections and normalization.

4.4 Task based MLP Layers

After the final layer of the Graph Transformer, the resulting node embeddings are processed through a Multi-Layer Perceptron (MLP) tailored to the specific learning task. This MLP generates the output predictions relevant to the target application. These outputs are subsequently used in the loss computation to guide the model's training process. The formal structure and definitions of these task-specific layers are detailed in the [Appendix B](#)

Chapter 5

Numerical Experiments

The effectiveness of the proposed Graph Transformer architecture is assessed using multiple benchmark datasets, namely ZINC (Irwin et al., 2012), CLUSTER, and PATTERN (Abbe, 2018), all under a constrained model size of roughly 500K parameters to ensure fair comparison.

Each dataset serves a different graph learning task and has unique characteristics:

- **ZINC:** This molecular graph dataset is utilized for property regression tasks. Due to the presence of detailed edge-level features (e.g., types of chemical bonds), the Graph Transformer variant with edge feature integration is applied. The dataset includes approximately 12K samples.
- **PATTERN and CLUSTER:** These datasets are designed for node classification and are synthetically generated using Stochastic Block Models (SBM). As they lack explicit edge features, the standard Graph Transformer without edge-specific processing is employed. PATTERN requires nodes to be categorized into two groups and contains around 14K graph instances, whereas CLUSTER involves six-class node labeling and comprises roughly 12K graphs.

The experimental procedure adheres to the benchmarking framework established by Dwivedi et al., 2023, utilizing PyTorch (Paszke, 2019) and DGL (Wang, 2019) libraries. The Graph Transformer is constructed with 10 stacked layers, each incorporating 8 attention heads. Hidden dimensions are configured arbitrarily to ensure the overall model complexity remains within a 500K parameter limit.

Model training employs a learning rate decay schedule, where the process terminates once the learning rate decreases to 1×10^{-6} . To ensure consistency and robustness, each configuration is executed using four different random seeds, with the final results reported as the mean and standard deviation across these runs.

5.1 Addressing Questions

As shown in Table 5.1, the Graph Transformer consistently outperforms both Graph Convolutional Networks (GCNs) (Kipf and Welling, 2016) and GATs (Veličković et al., 2017) across all three benchmark datasets. GCNs operate by aggregating features from neighboring nodes through spectral convolutions, while GATs introduce attention mechanisms to assign different weights to neighbors during message passing. Although GATs account for sparse graph structures, they lack positional encodings, which restricts their ability to model nuanced structural information. The Graph Transformer addresses this limitation

by incorporating both graph sparsity and positional encoding, resulting in more expressive and robust representations.

However, the model does not surpass the performance of Gated Graph Convolutional Networks (GatedGCNs) (Bresson and Laurent, 2017), which integrate gating mechanisms into message passing and achieve superior results across all datasets. The authors of this work highlights that the primary goal of this work was not to reach state-of-the-art benchmarks but to present a simplified and generalizable extension of the transformer framework for graph data.

Model	ZINC	CLUSTER	PATTERN
GCN	0.367 ± 0.011	68.498 ± 0.976	71.892 ± 0.334
GAT	0.384 ± 0.007	70.587 ± 0.447	78.271 ± 0.186
GatedGCN	0.214 ± 0.013	76.082 ± 0.196	86.508 ± 0.085
GT RESULTS			
GT	0.226 ± 0.014	73.169 ± 0.622	84.808 ± 0.068

TABLE 5.1: For ZINC, lower scores indicate better performance. For PATTERN and CLUSTER, higher scores indicate better performance

Several important points can be drawn from the open questions explored earlier in the related work section. First, does local attention beat full attention, and should graph sparsity be used in every layer? The ablation study (in table 5.2) compared local attention (just nearby nodes) to full attention across datasets. Results show local attention with graph sparsity works better consistently. This means graph data has useful structure that should be built into the layers for better results. Plus, local attention is faster, while full attention takes more computing time.

Dataset	Graph	Test Perf. $\pm$ s.d.	Train Perf. $\pm$ s.d.	#Epoch	Epoch/Total
ZINC	Sparse	0.226 ± 0.014	0.059 ± 0.011	287.50	27.78 s/2.25hr
	Full	0.598 ± 0.049	0.339 ± 0.123	273.50	45.26 s/3.50hr
CLUSTER	Sparse	73.169 ± 0.622	86.585 ± 0.905	126.50	201.06 s/7.20hr
	Full	27.121 ± 8.471	27.192 ± 8.8485	133.75	552.06 s/20.72hr
PATTERN	Sparse	84.808 ± 0.068	86.559 ± 0.116	145.25	309.95 s/12.67hr
	Full	54.941 ± 3.739	54.915 ± 3.769	117.75	683.53 s/22.77hr

TABLE 5.2: Comparison of Sparse and Full graph

Second, an important question concerns which type of positional encoding yields the best results. Weisfeiler-Lehman Positional Encoding (WLPE), introduced in prior work such as GraphBERT (Zhang et al., 2020), encodes structural similarity between nodes based on the Weisfeiler-Lehman test (Niepert, Ahmed, and Kutzkov, 2016). However, the ablation results in Table 5.3 indicate that Laplacian Positional Encoding (LapPE) provides better performance. LPE effectively distinguishes nodes by encoding proximity—nodes that are closer

exhibit more similar features—thereby capturing the underlying graph structure more faithfully. In contrast, WLPE tends to overfit, leading to improved training accuracy but reduced generalization.

Dataset	PE	Sparse Graph			
		Test Perf. $\pm$ s.d.	Train Perf. $\pm$ s.d.	#Epoch	Epoch/Total
ZINC	X	0.264 ± 0.008	0.048 ± 0.006	321.50	28.01 s/2.52hr
	L	0.226 ± 0.014	0.059 ± 0.011	287.50	27.78 s/2.25hr
	W	0.267 ± 0.012	0.059 ± 0.010	263.25	27.04 s/2.00hr
CLUSTER	X	72.139 ± 0.405	85.857 ± 0.555	121.75	200.85 s/6.88hr
	L	73.169 ± 0.622	86.585 ± 0.905	126.50	201.06 s/7.20hr
	W	70.790 ± 0.537	86.829 ± 0.745	119.00	196.41 s/6.69hr
PATTERN	X	83.949 ± 0.303	83.864 ± 0.489	236.50	299.54 s/19.71hr
	L	84.808 ± 0.068	86.559 ± 0.116	145.25	309.95 s/12.67hr
	W	75.489 ± 0.216	97.028 ± 0.104	109.25	310.11 s/9.73hr

TABLE 5.3: Comparison of Graph Transformer using different PEs Notations
X: No PE; L: LapPE; W: WLPE

5.2 Limitations

Calculating the eigenvectors of the Laplacian matrix, which are crucial for positional encodings in graph transformer models, becomes computationally challenging as graph size increases. This is because the process of eigen decomposition is both time-consuming and resource-intensive for large-scale graphs. **Computational Cost:** The typical time complexity for eigenvector computation is $\mathcal{O}(n^3)$, where n represents the number of nodes—making it unsuitable for very large graphs. **Memory Overhead:** Handling and storing these large matrices also demands significant memory, often surpassing the limits of conventional hardware.

Chapter 6

Conclusion

This work introduces a general-purpose Graph Transformer architecture, successfully adapting transformer models to graph-structured data. While the proposed Graph Transformer shows promising results and outperforms baseline models like across various benchmarks, it remains a straightforward architecture, focusing on generalization rather than state-of-the-art performance on specific datasets. Key enhancements, such as Laplacian eigenvectors for positional encoding and graph sparsity, significantly boost performance. Additionally, the extension to include edge features enables the model to approach the performance of top models like GatedGCN, particularly in datasets with rich edge attributes.

Although the model sets a strong foundation for future research in graph attention models, its performance could be further improved by incorporating domain-specific inductive biases. Future work should focus on refining the Graph Transformer by learning from advanced transformer architectures and exploring ways to improve scalability for large graphs and adaptability for heterogeneous domains. This research paves the way for future advancements in Graph Transformers, with the goal of achieving state-of-the-art results in graph-based tasks.

Appendix A

Transformer

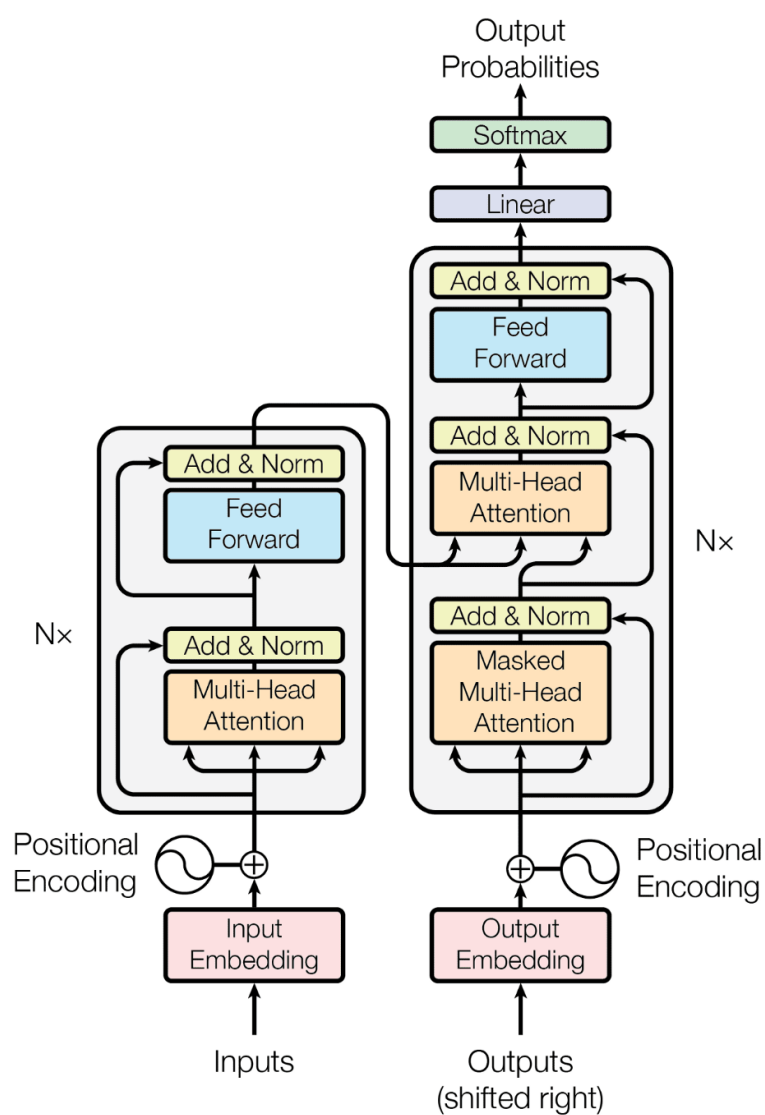


FIGURE A.1: Transformer Architecture

Appendix B

Task based MLP layer equations

For graph prediction tasks, the graph-level feature vector y_G is computed by averaging the final layer node features of the graph, resulting in a d -dimensional vector:

$$y_G = \frac{1}{V} \sum_{i=0}^V h_i^L$$

This feature vector is then fed into an MLP to obtain un-normalized prediction scores for each class, where y_{pred} is calculated as:

$$y_{\text{pred}} = P \text{ReLU}(Q y_G)$$

Here, $P \in \mathbb{R}^{d \times C}$ and $Q \in \mathbb{R}^{d \times d}$, with C representing the number of classes. For tasks such as single-target graph regression (e.g., in ZINC), $C = 1$, and the L1-loss between predicted and actual values is minimized during training.

For node prediction tasks, each node's feature vector is passed through an MLP to compute the un-normalized prediction scores $y_{i,\text{pred}}$ for each class:

$$y_{i,\text{pred}} = P \text{ReLU}(Q h_i^L)$$

Where $P \in \mathbb{R}^{d \times C}$ and $Q \in \mathbb{R}^{d \times d}$. The cross-entropy loss, weighted by the inverse of the class size, is used during training.

These task-specific layers can be adapted depending on the dataset or the type of prediction required. For instance, Graph Transformer outputs (Figure 4.1, Right) could be used for edge prediction tasks, and the MLP layers would be defined similarly to those used in node prediction.

Bibliography

- Abbe, Emmanuel (2018). “Community detection and stochastic block models: recent developments”. In: *Journal of Machine Learning Research* 18.177, pp. 1–86.
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473*.
- Belkin, Mikhail and Partha Niyogi (2003). “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6, pp. 1373–1396.
- Bresson, Xavier and Thomas Laurent (2017). “Residual gated graph convnets”. In: *arXiv preprint arXiv:1711.07553*.
- Chami, Ines et al. (2020). “Low-dimensional hyperbolic knowledge graph embeddings”. In: *arXiv preprint arXiv:2005.00545*.
- Cranmer, Miles D et al. (2019). “Learning symbolic physics with graph networks”. In: *arXiv preprint arXiv:1909.05862*.
- Devlin, Jacob et al. (2019). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186.
- Dwivedi, Vijay Prakash et al. (2023). “Benchmarking graph neural networks”. In: *Journal of Machine Learning Research* 24.43, pp. 1–48.
- Gilmer, Justin et al. (2017). “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR, pp. 1263–1272.
- Hu, Ziniu et al. (2020). “Heterogeneous graph transformer”. In: *Proceedings of the web conference 2020*, pp. 2704–2710.
- Irwin, John J et al. (2012). “ZINC: a free tool to discover chemistry for biology”. In: *Journal of chemical information and modeling* 52.7, pp. 1757–1768.
- Kipf, Thomas N and Max Welling (2016). “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907*.
- Li, Yuan et al. (2019). *Graph Transformer*. URL: <https://openreview.net/forum?id=HJei-2RcK7>.
- Monti, Federico et al. (2019). “Fake news detection on social media using geometric deep learning”. In: *arXiv preprint arXiv:1902.06673*.
- Nguyen, DQ, T Dinh Nguyen, and D Phung (2019). “Universal self-attention network for graph classification”. In: *arXiv preprint arXiv:1909.11855*.
- Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov (2016). “Learning convolutional neural networks for graphs”. In: *International conference on machine learning*. PMLR, pp. 2014–2023.
- Paszke, A (2019). “Pytorch: An imperative style, high-performance deep learning library”. In: *arXiv preprint arXiv:1912.01703*.
- Radford, Alec et al. (2018). “Improving language understanding by generative pre-training”. In.

- Sanchez-Gonzalez, Alvaro et al. (2020). "Learning to simulate complex physics with graph networks". In: *International conference on machine learning*. PMLR, pp. 8459–8468.
- Schlichtkrull, Michael et al. (2018). "Modeling relational data with graph convolutional networks". In: *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*. Springer, pp. 593–607.
- Srinivasan, Balasubramaniam and Bruno Ribeiro (2019). "On the equivalence between positional node embeddings and structural graph representations". In: *arXiv preprint arXiv:1910.00452*.
- Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.
- Veličković, Petar et al. (2017). "Graph attention networks". In: *arXiv preprint arXiv:1710.10903*.
- Wang, Minjie Yu (2019). "Deep graph library: Towards efficient and scalable deep learning on graphs". In: *ICLR workshop on representation learning on graphs and manifolds*.
- You, Jiaxuan, Rex Ying, and Jure Leskovec (2019). "Position-aware graph neural networks". In: *International conference on machine learning*. PMLR, pp. 7134–7143.
- Yun, Seongjun et al. (2019). "Graph transformer networks". In: *Advances in neural information processing systems* 32.
- Zhang, Jiawei et al. (2020). "Graph-bert: Only attention is needed for learning graph representations". In: *arXiv preprint arXiv:2001.05140*.