Bringing k-MIP Attention to Graph Transformers

Thesis for the MSc in Advanced Computer Science

Jonas De Schouwer

University of Oxford

September 2024

Abstract

Graph machine learning has become an essential tool for analysing relational data, which is prevalent across domains like social networks, molecular biology, and recommendation systems. While graph Transformers overcome key limitations of traditional message-passing neural networks, their computational overhead can be prohibitive for large graphs. To address this issue, we introduce the k-Maximum Inner Product Graph Transformer (k-MIP-GT), a novel and versatile graph Transformer architecture that leverages the k-Maximum Inner Product (k-MIP) self-attention mechanism within the GraphGPS framework.

The k-MIP-GT balances scalability and performance, delivering competitive results on both small and medium-sized graph datasets, while its attention mechanism is two orders of magnitude faster than full attention. We establish theoretical guarantees for its universal approximation properties, extending analogous results for both standard and graph Transformers. Additionally, we investigate various properties of the k-MIP-GT, including the influence of two key hyperparameters, the degree to which it approximates full self-attention, and the characteristics of its generated attention graphs.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I express my gratitude to my supervisors, Haitz Sáez de Ocáriz Borde, Prof. Xiaowen Dong, and Prof. Michael Bronstein, for their invaluable guidance and support throughout the course of this dissertation.

A special thanks goes to Prof. Xiaowen Dong for providing me with access to the Advanced Research Computing (ARC) service, which was essential for the computational aspects of this work. I am also incredibly grateful to the technical support staff of ARC for their unwavering assistance. Their guidance through long days (and nights) made a significant difference.

Finally, I would like to thank my family and friends for their constant support and understanding. Their encouragement has been a source of strength throughout this journey.

1 | Introduction

Relational data is ubiquitous across numerous domains, ranging from social networks [1, 2] and recommendation systems [3, 4] to materials science [5, 6], molecular biology [7, 8], and weather forecasting [9, 10]. Graphs, which are abstractions consisting of entities (nodes) and the relationships between them (edges), are often the most natural way to represent such data. Following the many recent successes in deep learning, graph machine learning has emerged as a powerful tool for analysing graph-structured data.

Traditional approaches to graph machine learning typically follow the message-passing paradigm, where information is propagated locally between neighbouring nodes in the given network. Despite its effectiveness, the limitations of this paradigm are becoming increasingly apparent as the field matures. These include oversmoothing [11], oversquashing [12] and limited expressivity [13, 14, 15], which may lead to difficulties in capturing long-range dependencies [16]. Graph Transformers have recently emerged as a promising alternative, avoiding these limitations by allowing information to be exchanged between unconnected nodes. Nonetheless, it remains an open question how to adapt the traditional (sequence) Transformer architecture [17] to effectively handle graph-structured data, while being scalable to large graphs.

To tackle this challenge, various approaches have been proposed in the literature. The most prominent strategies include linearised attention mechanisms [18, 19, 20], manually engineered attention patterns [21], and graph subsampling methods [22, 23]. Each of these, however, comes with its own trade-offs. Linearised attention mechanisms, while more efficient, introduce approximation errors that lead to degraded performance compared to full softmax attention [24]. Manually engineered attention patterns, on the other hand, can be overly rigid, inhibiting the model's ability to capture important relationships between nodes. Finally, graph subsampling methods are unable to capture long-range dependencies in the graph, as they do not permit information flow between distant nodes.

To address these limitations, this dissertation introduces the k-Maximum Inner Product Graph Transformer (k-MIP-GT), a novel graph Transformer architecture based on the k-Maximum Inner Product (k-MIP) self-attention mechanism [25, 26]. k-MIP self-attention dynamically selects the k most influential keys for each query based on the intermediate activations, achieving scalability while avoiding the drawbacks of linearisation, rigid attention patterns and graph subsampling.

Our main contributions are as follows:

- We introduce the k-MIP Graph Transformer, an efficient and versatile architecture that supports a wide range of graph learning tasks.
- We propose the use of symbolic matrices to drastically improve the computational efficiency and memory footprint of the k-MIP self-attention mechanism.
- We evaluate the performance of the k-MIP-GT across various graph learning tasks and achieve results competitive with state-of-the-art graph Transformers on both small (<2K nodes) and medium-sized (2K-500K nodes) graph datasets.
- We provide universal approximation guarantees for our k-MIP (Graph) Transformer, analogous to those previously established for standard Transformers [27] and graph Transformers [21, 28].
- We analyse five properties of the k-MIP Graph Transformer:
  (1) the efficiency of the k-MIP attention mechanism,
  (2) the influence of the hyperparameter k on performance and efficiency,
  (3) the robustness to low dimensionalities of the key-query space,
  (4) the degree to which it approximates the full self-attention mechanism,
  (5) the characteristics of the attention graphs it generates.

Overall, this dissertation is the first work to bring the k-MIP self-attention mechanism to the graph domain and to propose the use of symbolic matrices to further enhance its scalability. The resulting k-MIP Graph Transformer achieves a speedup of two orders of magnitude compared to full attention, while maintaining competitive performance across a wide range of graph datasets.

The remainder of this dissertation is structured as follows. In chapter 2, we review the relevant literature and contextualize our work. Chapter 3 introduces the notation and concepts necessary for understanding the rest of the dissertation. In chapter 4, we present the k-MIP Graph Transformer method in detail. Chapter 5 provides theoretical guarantees on the expressive power of the k-MIP Graph Transformer. Chapter 6 presents all experimental results, including the evaluation of the k-MIP Graph Transformer on various graph learning tasks and the analysis of the architecture's properties. Finally, Chapter 7 concludes the dissertation and discusses both the limitations of our work and potential avenues for future research.

To conduct this dissertation, we developed two codebases. One codebase served for the integrated experiments where the full k-MIP Graph Transformer is trained on various datasets, i.e. the experiments in sections 6.1, 6.4, 6.5, 6.6, and 6.8. This codebase is available at

`https://anonymous.4open.science/r/k-MIP-Graph-Transformer`

The other codebase served for the experiments where the k-MIP attention mechanism is studied and benchmarked in isolation, i.e. the experiments in sections 6.2, 6.3 and 6.7. This codebase is available at

`https://anonymous.4open.science/r/Efficient-k-MIP-Attention`

2 | Background

In this chapter we review the relevant literature to contextualise our work. We begin by introducing the field of graph machine learning, highlighting the challenges faced by traditional graph neural networks and how graph Transformers can overcome them. Following this, we explore efficient Transformers in the sequence domain, which have been developed to address the computational limitations of the standard Transformer architecture. Understanding these advancements is essential, as they have inspired many of the methods and ideas used in graph Transformers. Next, we provide an overview of the most influential architectures in the field of graph Transformers, with particular attention to those designed to scale efficiently to large graphs. Finally, we discuss prior work on the expressive power of (graph) neural networks and (graph) Transformers, to provide context and a theoretical foundation for the expressivity guarantees we will present in chapter 5.

2.1 Graph Machine Learning

Graph machine learning is a rapidly growing subfield of machine learning focused on algorithms for graph-structured data. Graphs are a data structure consisting of entities (nodes) and the relationships between them (edges). Because of their versatility for modelling interconnected systems, they are widely used across various fields. Hence, the ability to perform machine learning using these graphs as a geometric prior is of great value, as it enables the extraction of valuable insights from these structures. Some examples include predicting molecular properties for drug discovery [29], recommending friends in social networks [30], and detecting fraudulent activities in financial systems [31]. The most common tasks in graph machine learning include node classification or regression, link prediction, graph classification or regression, and graph generation.

One of the key challenges in graph machine learning is handling the inherent invariance of the graph domain with respect to permutations of the node ordering

(formally defined in section 3.2). To do this, the majority of traditional graph neural networks (GNNs) follow the message-passing paradigm [32], which iteratively updates node features by aggregating information from neighbouring nodes. Many popular GNN architectures fall under this framework, including Graph Convolutional Network (GCN) [33], Graph Attention Network (GAT) [34], and Graph Isomorphism Network (GIN) [14].

The message-passing paradigm assumes that an input graph is given and that it is suited for learning the task at hand. A badly conditioned graph, however, can lead to suboptimal performance that cannot simply be solved with better GNN models. The problems that the literature focuses on most are *oversmoothing* and *oversquashing*. Oversmoothing [11] refers to the phenomenon where, as the number of layers in a GNN increases, the node representations tend to become indistinguishable from each other, losing the ability to capture meaningful differences. On the other hand, oversquashing [12] occurs when the aggregation of messages along a path in a graph compresses an exponentially growing amount of information into fixed-size vectors, which limits the model's ability to effectively propagate signals over long distances in the graph.

Studies [35] and [36] show that the graph topology is a deciding factor for both oversmoothing and oversquashing. Hence, the key realization for solving both problems is that one does not need to restrict the information flow to the input graph. Graph Transformers [37, 38] have emerged as a promising alternative to traditional GNNs that addresses these issues, precisely by allowing information to be exchanged between unconnected nodes. We will review graph Transformers in more detail in section 2.3.

2.2 Efficient Transformers

As graphs can contain an arbitrarily large number of nodes, efficiency and scalability are critical concerns for many graph Transformer architectures. To address these challenges, numerous graph Transformers draw inspiration from efficient Transform-

ers originally designed for sequence data, with some even integrating these sequence-based models as key components. Therefore, it is important to first review the most influential techniques and ideas that have been developed to enhance the scalability of the Transformer architecture in the sequence domain. This exploration will provide valuable insights into how similar challenges are tackled within graph Transformers.

The Transformer architecture, first introduced by [17] for sequence transduction tasks, marked a significant advancement in the field of deep learning. Besides natural language processing, it has gained popularity in various machine learning communities, including computer vision [39], speech processing [40], and genomic data analysis [41]. The architecture's key innovation lies in its reliance on the multi-head self-attention mechanism (discussed in detail in section 3.5), while it dispenses with recurrence and convolutions entirely. This allows it to directly model relationships between distant tokens in a sequence, making it possible to capture long-range dependencies much more effectively than traditional sequential methods like RNNs [42] and LSTMs [43]. Further, the simultaneous processing of all tokens in a sequence allows for highly parallelizable computations, which can be efficiently implemented on modern computing hardware like GPUs and TPUs.

The original Transformer architecture, however, has a quadratic time and memory complexity with respect to the sequence length. This makes it unsuitable for processing very long sequences, often referred to as "large context windows" in natural language processing. This limitation has led to the development of several methods to make the Transformer architecture more scalable. In what follows, we follow the taxonomy proposed in [44] to present the most influential methods for our work. This taxonomy presents different non-exclusive techniques used to reduce the computational complexity of the Transformer architecture, and assigns each efficient Transformer method to one or more of the following four categories (simplified for clarity):

(1) kernels and low-rank factorization,

(2) fixed, factorized or random patterns,

(3) sparse attention, and

(4) learnable patterns.

The first category involves methods that approximate the self-attention mechanism through low-rank approximations of the attention matrix and/or kernelization to reduce complexity. A prominent example of this category is the Linformer [45], which projects the $N \times d$ dimensional keys and values to $k \times d$ dimensions before attention, which provably leads to an approximation of the full attention mechanism. Further, there is the Performer [46], which employs a kernelized attention mechanism and avoids computing the full attention matrix through an Orthogonal Random Features approximation [47]. Finally, the Linear Transformer [48] removes the softmax operation from the self-attention mechanism, which allows for a factorization of the mechanism that has a linear complexity with respect to the sequence length.

The second category involves methods that manually engineer which tokens can attend to one another. Sparse Transformer [49], for example, dedicates half of its attention heads to attention on nearby tokens, while the other half attends to tokens in a strided pattern. Longformer [50] is a variation of this that introduces the "Dilated Sliding Window" attention pattern. ETC [51], on the other hand, introduces virtual "global tokens" and has an attention mechanism that consists of four components: global-to-global, global-to-local, local-to-global, and local-to-local. Finally, the BigBird architecture [52] uses a combination of three attention patterns: global attention w.r.t. a number of global tokens, sliding window attention w.r.t. nearby tokens and random attention w.r.t. a number of random tokens.

The third category involves methods that sparsely activate a subset of the parameters, which generally improves the parameter to FLOP ratio [44]. This mostly includes Mixture-of-Experts models like GShard [53], Switch Transformer [54], and GLaM [55]. However, as these methods are not directly related to our work, we will not discuss them further.

The fourth category involves methods that adaptively learn which tokens to attend to. Since our own method belongs to this category, we will provide a more in-depth

discussion of the relevant prior work. Reformer [56] introduces a mechanism based on locality sensitive hashing. In this approach, each key and query is first projected into a space with dimension $b/2$ using a random matrix and then hashed into $2^{b/2}$ angular buckets. The attention mechanism is subsequently applied within each bucket separately. Similarly, the Routing Transformer [57] clusters normalized key and query vectors at each self-attention step and limits attention computation to tokens within each cluster. These clusters are determined by a spherical k-means algorithm whose centroids are updated iteratively. Clusterformer [58] works analogously to Routing Transformer, but employs Euclidean k-means on the unnormalized key and query vectors to form clusters. The Sinkhorn Transformer [59] takes a different approach, leveraging the Sinkhorn algorithm to approximate full attention. This algorithm creates a differentiable sorting network for each head that assigns a relevant key block to each query block and then performs attention between those corresponding blocks. Clustered Attention [60] uses a more hierarchical approach, clustering the queries and performing attention only for the cluster centroids (but w.r.t. all keys). Each output token is assigned the attention result of the closest cluster centroid, reducing the computational cost while preserving the relevance of attended information.

In this work, we employ the self-attention mechanism that is used in Explicit Sparse Transformer [26] and KVT [25]: for each query, we perform attention only for the k keys that have the maximum inner product with that query, where k is a hyperparameter. From now onwards, we will refer to this approach as *k-MIP attention*. This was shown to be effective in the vision domain in [25], where introducing k-MIP attention led to improved performance in various vision Transformers at a comparable computational cost.

To the best of our knowledge, we are the first to bring k-MIP attention to the graph domain. While k-MIP attention is already more efficient than full attention for large numbers of tokens, we employ symbolic matrices to achieve a further speedup of an order of magnitude, an optimisation of k-MIP attention that has not been explored in the literature before.

For a more extensive overview of efficient Transformer architectures, we refer the reader to [44].

2.3 Graph Transformers

Inspired by their success in the sequence domain, researchers have started leveraging the Transformer architecture for graph-structured data [37, 38, 61]. The key idea is to treat the nodes in a graph as tokens in a sequence. This approach has the double advantage that it is highly parallelizable and that it alleviates the problems highlighted in section 2.1 – in particular, oversmoothing and oversquashing – by allowing attention between unconnected nodes.

While there are some graph Transformers that treat the edges as tokens, such as TokenGT [62] and the edge-wise LPE Transformer from [28], these methods incur a heavy computational cost because the number of edges in a graph is often much larger than the number of nodes. Therefore, they are unsuited for scaling to large graphs.

Most graph Transformers that treat nodes as tokens follow the global all-to-all attention pattern of the original Transformer architecture [17]. Examples of this include the original Graph Transformer [38], Graphormer [37], Transformer-M [63], GRPE [64], and SAN [28]. One advantage of this approach is that materialising the full attention matrix allows the introduction of edge information into the attention mechanism, which is done by all of the mentioned examples except from SAN, and has been shown to improve expressivity [65]. However, this approach has a quadratic time and memory complexity with respect to the sequence length, making it unsuitable for large graphs, which potentially contain millions of nodes. As a result, the graph Transformers mentioned above have not been demonstrated on graph datasets with more than a few hundred nodes per graph. To the best of our knowledge, the largest graph dataset on which a graph Transformer with a full attention mechanism has been demonstrated is the MalNet-Tiny dataset [66, 67], which contains graphs with up to 5000 nodes and has been used to train the GPS+Transformer model [67].

In what follows, we will focus on methods that have been proposed to make the self-attention mechanism more efficient, but exclude those methods that only perform local attention on the neighbourhood of each node, such as Graph Attention Network (GAT) [34]. We can divide most of those methods into the same four categories as the efficient Transformers in the previous section, according to the type of their efficient attention mechanism. We include a fifth category for methods that obtain scalability through other means, and a sixth for more general frameworks that are flexible with respect to the attention mechanism used. To summarize, the categories are:

(1) kernels and low-rank factorization,

(2) fixed, factorized or random patterns,

(3) sparse attention,

(4) learnable patterns,

(5) graph subsampling, and

(6) frameworks flexible with respect to the attention mechanism used.

The first class of mechanisms used in efficient graph Transformers are the linearised attention methods. This class contains the Nodeformer [18], which reframes the self-attention mechanism using a kernelized Gumbel-Softmax operator [68] and achieves a linear complexity through a Positive Random feature approximation, similar to the mechanism adopted by Performer [46] in the sequence domain. Some other examples are the Difformer and SGFormer [19, 20], which both use a linearised self-attention mechanism without a softmax operation, such that the attention matrix can be factorized. However, these linearised attention mechanisms often demonstrate worse performance than full softmax attention [24].

The second category involves methods that manually engineer which nodes can attend to one another. The main example of this is Exphormer [21], which performs attention on (1) the neighbours of each node, (2) a small number of virtual global nodes, and (3) a precomputed random expander graph. They obtained state-of-the-art performance on a variety of datasets, and we will hence use Exphormer as a

baseline in our experiments. While the approaches in this category may be efficient, they are potentially suboptimal because they engineer the attention patterns based on solely the input graph, rather than choosing which nodes to attend to based on the data, including node and edge features at each layer.

The third category involves methods that sparsely activate a subset of the parameters, which in practice mostly includes Mixture-of-Experts models. Limited work has been done in this area for graph Transformers, with only a few examples of applied integrated methods [22, 23]. However, as this technique is orthogonal to the other three categories and has shown to be effective in the sequence domain, it is a promising research direction for future work.

The fourth category involves methods that adaptively learn which nodes to attend to. This has the potential advantage of focusing attention on the most relevant parts of the graph based on the data (including intermediate activations), thus filtering out noise and potentially improving the model's efficiency. GOAT [69] is an example of such a method. It proceeds in a similar vein as Clusterformer [58] (an efficient sequence Transformer reviewed in the previous section), but clusters the keys with Euclidean k-means rather than the queries. It keeps track of the keys and value vectors corresponding to each cluster centroid, updates these with an exponential moving average, and only performs attention with respect to the centroid keys (but for all queries). Beyond this, only limited work has been done in this category [70]. We believe that the potential in this area has been largely overlooked in the current literature, and the novel method proposed in this dissertation offers a significant advancement in this field.

The fifth category consists of graph Transformers that do employ the vanilla self-attention mechanism, but leverage graph subsampling to be scalable to larger graphs. One such example is Gophormer [71], which samples 'ego-graphs' (i.e. the induced graph of a node and its $\leq D$-hop neighbours) from a large graph and performs attention on them in the style of GraphSAGE [72]. NAGphormer [73] uses a similar technique, but instead of sampling ego-graphs, it constructs a sequence for each node based on an aggregation of the features of the node's k-hop neighbourhood,

for $k \in \{1, \ldots, K\}$. While these approaches can be efficient, they are unable to capture long-range dependencies in the graph, as they do not permit information flow between distant nodes.

Finally, there are some graph Transformer frameworks that are designed with modularity in mind, allowing for the use of various attention mechanisms, and can thus fit into each of the first four categories. GraphGPS [67] is the most prominent example of such frameworks, which we will explain in more detail in section 3.7. It has been shown to be effective for building performant graph Transformers in both the original GraphGPS paper [67] and in derivative works, which include Exphormer [21] and GPS++ [74]. For this reason, and because it allows an easy integration of our k-MIP self-attention mechanism, we use GraphGPS as the basis for our own graph Transformer.

To the best of our knowledge, we are the first to bring k-MIP self-attention to the graph domain.

2.4 Expressive Power of GNNs, Transformers and Graph Transformers

The expressive power of a parametrized model refers to the range of functions it can represent. Understanding the expressiveness of a model is crucial for understanding its limitations and for designing models that can solve a given task. For feedforward neural networks, the research towards this culminated in the universal approximation theorem [75], which states a feedforward neural network with a single hidden layer, a nonpolynomial activation function and a sufficient number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary precision.

Similarly, a more recent paper [27] has shown that full-attention Transformer networks of constant width and sufficient depth can approximate any continuous sequence-to-sequence function to arbitrary precision. In this dissertation, we will prove a completely analogous result for sequence-to-sequence Transformers leveraging the

k-MIP self-attention mechanism, showing that the class of Transformers based on this mechanism is equally expressive as the class of full-attention Transformers.

For models that learn from graph-structured data, the matter of expressive power is a lot more intricate, as a model's ability to incorporate node features, graph structure, edge weights and edge features all contribute to its overall expressiveness. Although there are numerous ways to quantify the expressive power of such methods, important upper bounds have been derived on the expressive power of MPNNs by studying their ability to distinguish non-isomorphic graphs. In particular, it has been shown that traditional MPNNs are at most as powerful as the Weisfeiler-Lehman (1-WL) test for detecting graph isomorphisms [14, 13]. As a consequence, MPNNs are unable to represent any function that assigns different values to graphs that the 1-WL test cannot distinguish.

A lot of work has gone into overcoming this limitation, leading to the development of more expressive models like higher-order GNNs [13], as well as augmentations to the input features leading to higher expressive power. In early works, such augmentations took the form of unique and/or random node identifiers [15, 76], which unfortunately break the permutation invariance or equivariance of the GNN. Hence, more recent works have employed positional encodings based on eigenvectors of the adjacency matrix or Laplacian of the graph [77, 78], node and distance metrics [79], substructure counts [80], or random walks [77]. Many of these have been shown to lead to an expressive power beyond the 1-WL test.

Incorporating positional encodings into the Weisfeiler-Lehman test leads to a preorder of WL test variations, where more discriminative encodings lead to a higher power for graph distinguishability. This preorder has been formalized in terms of the Structural Encoding Enhanced Global Weisfeiler-Lehman Test (SEG-WL test) from [65].

In their work, [65] also characterise the expressive power of various graph Transformer architectures from section 2.3 in terms of a SEG-WL test, including the original graph Transformer [38], Graphormer [37], SAN [28], Gophormer [71], and SAT [81]. In this dissertation, we extend their work by showing that the expressive

power of the GraphGPS framework – including our own k-MIP-GT, Exphormer [21], and GPS++ [74] – is upper bounded by a SEG-WL test with a structural encoding scheme that is determined by the input node features, the input edge features, and the positional encodings used in the model. In particular, we show that in the usual 1-WL setting, where all node and edge features are identical and there are no positional encodings, the GraphGPS framework cannot distinguish graphs that the 1-WL test cannot distinguish. We use this insight to advocate for the use of expressive positional encodings.

When sufficiently expressive positional encodings are used, however, many graph Transformers can leverage the universal approximation property of sequence Transformers to approximate any continuous function on graphs to arbitrary precision. Such a universal approximation result has been proven for SAN [28] and Exphormer [21] under the assumption that a maximally expressive positional encoding (the padded adjacency matrix) is used. In this dissertation, we will prove a completely analogous result for the k-MIP-GT, showing that no expressivity is lost by using the k-MIP self-attention mechanism in graph Transformers.

3 | Preliminaries

In this chapter, we establish the foundational concepts and notations necessary for understanding the rest of this dissertation. We begin by introducing the formal definition of graphs and the associated terminology, which serves as the basis for all subsequent discussions.

Next, we cover the important concepts of equivariance and invariance, both of which play a critical role in ensuring that models respect the symmetries inherent in graph data. We also provide an overview of Message-Passing Neural Networks (MPNNs), a widely adopted framework in graph-based learning, and describe how they are employed in this work.

Additionally, we explain the mechanics of Multi-Layer Perceptrons (MLPs) and the self-attention mechanism, both of which will be used internally in our k-MIP-GT. To help graph Transformers incorporate graph structure, we also discuss the use of positional and structural encodings.

Following this, we introduce the GraphGPS framework, which combines MPNNs and global attention layers to form a flexible and powerful graph Transformer architecture. Finally, we conclude with a discussion on the Weisfeiler-Lehman (WL) test and the Structural Encoding Enhanced Global Weisfeiler-Lehman (SEG-WL) test, which provide the theoretical foundation for analysing the expressive power of our models later in this dissertation.

3.1 Graphs

A *graph* is a tuple $G = (V, E, \boldsymbol{X}, \boldsymbol{E}, \boldsymbol{A})$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$ is a node feature matrix, $\boldsymbol{E} \in \mathbb{R}^{|E| \times d}$ is an edge feature matrix and $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$ is the weighted adjacency matrix. $e = (v_i, v_j) \in E$ encodes that there is an edge that originates from v_i and terminates at v_j.

We denote the number of vertices in G by $N = |V|$ and the number of edges by

$M = |E|$. A graph is *undirected* if $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$, otherwise it is *directed*. The *in-degree* $d_{in}(v)$ of a vertex $v \in V$ is the number of edges terminating in v, while the *out-degree* $d_{out}(v)$ is the number of edges originating from v. If G is undirected, then the *degree* of a vertex is $d(v) = d_{in}(v) = d_{out}(v)$. The *in-neighbourhood* $\mathcal{N}_{in}(v)$ of a vertex v is the set of vertices that have an outgoing edge terminating at v, while the *out-neighbourhood* $\mathcal{N}_{out}(v)$ is the set of vertices that have an incident edge originating from v. If G is undirected, then the *neighbourhood* of a vertex v is $\mathcal{N}(v) = \mathcal{N}_{in}(v) = \mathcal{N}_{out}(v)$.

When dealing with matrices, we denote the i'th row of a matrix $\boldsymbol{M}$ by $\boldsymbol{M}_i$. As such, the *node feature vector* associated with node v_i is $\boldsymbol{x}_i = \boldsymbol{X}_i \in \mathbb{R}^d$. Likewise, the *edge feature vector* associated with edge e_i is $\boldsymbol{e}_i = \boldsymbol{E}_i \in \mathbb{R}^d$.

3.2 Equivariance and Invariance

Two related concepts that will recur frequently throughout this dissertation are *equivariance* and *invariance*. We will first give a formal definition of both concepts, and then motivate their importance in the context of graph machine learning.

Definition 1. *Let $\mathcal{G}, *$ be a group, let $\mathcal{X}$ be a set, and denote by $[\mathcal{X}, \mathcal{X}]$ the class of mappings from $\mathcal{X}$ to $\mathcal{X}$. A* representation *of $\mathcal{G}$ on $\mathcal{X}$ is a mapping $\rho : \mathcal{G} \to [\mathcal{X}, \mathcal{X}]$ that satisfies [82]:*

1. *$\rho(\mathcal{G}), \circ$ is a group, where $\circ$ denotes the composition of mappings.*
2. *$\rho : \mathcal{G}, * \to \rho(\mathcal{G}), \circ$ is a group homomorphism.*

Definition 2. *Let $f : \mathcal{X} \to \mathcal{Y}$ be a mapping of sets, $\mathcal{G}, *$ a group and $\rho_1 : \mathcal{G} \to [\mathcal{X}, \mathcal{X}]$, $\rho_2 : \mathcal{G} \to [\mathcal{Y}, \mathcal{Y}]$ representations of $\mathcal{G}$ on $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then f is* equivariant *with respect to $\mathcal{G}$ if for all $g \in \mathcal{G}$ [82]*

$$f \circ \rho_1(g) = \rho_2(g) \circ f. \tag{3.1}$$

Similarly, f is invariant *with respect to $\mathcal{G}$ if for all $g \in \mathcal{G}$ [82]*

$$f \circ \rho_1(g) = f. \tag{3.2}$$

In the field of graph machine learning, there is a lot of symmetry in the data. In particular, if the indices of the nodes in a graph are permuted, it still represents the same graph. More formally, let $G = (V, E, \boldsymbol{X}, \boldsymbol{E}, \boldsymbol{A})$ be any graph, $\pi \in \mathrm{Sym(V)}$ be any permutation of its vertices with associated permutation matrix $\boldsymbol{P}$, and $\tau : E \to E$ be the operation that transforms an edge $e = (v_i, v_j)$ to $e' = (v_{\pi(i)}, v_{\pi(j)})$. Then

$$G' = (\pi(V), \tau(E), \boldsymbol{PX}, \boldsymbol{E}, \boldsymbol{PAP}^\top) \cong G \tag{3.3}$$

still represents the same graph.

It is desirable that graph machine learning models give the same output for isomorphic graphs, irrespective of their specific representation, i.e. that the models are equivariant (for node or edge-level tasks) or invariant (for graph-level tasks) w.r.t. permutations of the node indices. While this behaviour can be learnt from the data, there are $N!$ possible permutations of the node indices, making it incredibly difficult to learn this behaviour from data alone. Instead, one can expect better generalization by carefully designing the model architecture to incorporate this symmetry as a geometric prior [83]. The traditional approach to do this is to combine permutation-equivariant and permutation-invariant layers, such as the message-passing neural network layers described in the next section.

3.3 Message-Passing Neural Networks

A *Message-Passing Neural Network* (MPNN) is a class of graph neural networks that iteratively update node and edge features of a graph by aggregating information from neighbouring nodes and edges [32]. The framework we use in this dissertation assumes that initial node and edge features $\{\boldsymbol{x}_u^{(0)} \mid u \in V\}$ and $\{\boldsymbol{e}_{uv}^{(0)} \mid (u, v) \in E\}$ are given and that the MPNN updates these features as follows, for each $u \in V$,

$(u, v) \in E$ and $l \in \{1, \dots, L\}$:

$$\boldsymbol{x}_u^{(l)} = \texttt{UPDATE}_{\text{node}}\left(\boldsymbol{x}_u^{(l-1)}, \texttt{AGG}\left(\{\!\{(\boldsymbol{x}_r^{(l-1)}, \boldsymbol{e}_{rv}^{(l-1)}) \mid r \in \mathcal{N}_{in}(u)\}\!\}\right)\right), \tag{3.4}$$

$$\boldsymbol{e}_{uv}^{(l)} = \texttt{UPDATE}_{\text{edge}}\left(\boldsymbol{e}_{uv}^{(l-1)}, \boldsymbol{x}_u^{(l-1)}, \boldsymbol{x}_v^{(l-1)}\right), \tag{3.5}$$

where $\texttt{UPDATE}_{\text{node}}$, $\texttt{UPDATE}_{\text{edge}}$, and $\texttt{AGG}$ are instance-specific functions. The function $\texttt{AGG}$ is invariant to the order of the elements in the multiset, which leads to the MPNN layer being equivariant w.r.t. permutations of the node indices.

Note that this framework is more general than the traditional definition, which only updates node features [32]. However, we will allow the MPNN component of our method to act on edge features as well, as this has been shown to improve performance in practice [67].

The MPNNs that we consider in this work, both as baselines and as components within our method, are Graph Convolutional Network (GCN) [33], Graph Attention Network (GAT) [34], Graph Isomorphism Network with Edges (GINE) [84] and Gated Graph Convolutional Network (GatedGCN) [85].

3.4 Multi-Layer Perceptrons

A *Multi-Layer Perceptron* (MLP) is a feedforward neural network that consists of linear layers of the form $\boldsymbol{X} \mapsto \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}$ with non-linear activation functions in between.

In this project, we will almost exclusively be working with two-layer MLPs with ReLU activation functions that have the same input and output dimensionality. When the input is a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$, this can be written as

$$\texttt{MLP}(\boldsymbol{X}) = \texttt{ReLU}(\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}_n \boldsymbol{b}_1^\top)\boldsymbol{W}_2 + \mathbf{1}_n \boldsymbol{b}_2^\top, \tag{3.6}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times r}, \boldsymbol{b}_1 \in \mathbb{R}^r, \boldsymbol{W}_2 \in \mathbb{R}^{r \times d}, \boldsymbol{b}_2 \in \mathbb{R}^d$ are learnable weight matrices and bias vectors. Here, r is the hidden layer size of the MLP.

Note that the operation performed by an MLP is row-wise, where each row of the input matrix is transformed independently of the others.

3.5 Self-Attention

A single layer of self-attention is a function that takes a set of tokens $(\boldsymbol{x}_i)_{i=1}^N$ and returns a set of equal length with updated tokens $(\boldsymbol{x}'_i)_{i=1}^N$. In natural language processing, these tokens typically represent words or subwords, while in graph machine learning, the tokens correspond to node feature vectors.

One of the key properties of self-attention is its equivariance to permutations of the input tokens. This property is particularly well-suited for graph-based tasks, as graphs do not have a natural ordering of nodes. However, to enable the model to capture the structure of the graph, additional information is needed. Positional encodings can be appended to the node feature vectors, embedding information about the graph's structure into the input tokens, as we will discuss further in section 3.6.

In the rest of this section, we will describe the self-attention mechanism in more detail.

3.5.1 Single-head Self-Attention

Single-head self-attention (SHSA) is a function that transforms a set of tokens $(\boldsymbol{x}_i)_{i=1}^N$ into an updated set of tokens of the same length $(\boldsymbol{x}'_i)_{i=1}^N$ and is equivariant to permutations of the tokens. In its standard form, it is computed as follows.

First, the tokens $(\boldsymbol{x}_i)_{i=1}^N$ are linearly transformed into *queries* $\boldsymbol{q}_i \in \mathbb{R}^{d_{kq}}$, *keys* $\boldsymbol{k}_j \in \mathbb{R}^{d_{kq}}$ and *values* $\boldsymbol{v}_j \in \mathbb{R}^{d_{val}}$ using learnt weight matrices $\boldsymbol{W}_Q \in \mathbb{R}^{d \times d_{kq}}$, $\boldsymbol{W}_K \in \mathbb{R}^{d \times d_{kq}}$

and $\boldsymbol{W}_V \in \mathbb{R}^{d \times d_{val}}$ as follows:

$$\boldsymbol{q}_i = \boldsymbol{W}_Q^\top \boldsymbol{x}_i, \tag{3.7}$$

$$\boldsymbol{k}_j = \boldsymbol{W}_K^\top \boldsymbol{x}_j, \tag{3.8}$$

$$\boldsymbol{v}_j = \boldsymbol{W}_V^\top \boldsymbol{x}_j. \tag{3.9}$$

The most common choice for the hidden dimensions is $d_{kq} = d_{val} = d$, but this is by no means a requirement. Indeed, in section 6.5, we will demonstrate that our k-MIP-GT method is robust to low values of d_{kq}, improving the efficiency of the model.

Next, the *attention scores* a_{ij} are computed as a softmax of the dot product between the queries and keys, scaled by $\sqrt{d_{kq}}$. For each $i, j \in \{1, \dots, N\}$,

$$e_{ij} = \frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d_{kq}}}, \tag{3.10}$$

$$a_{ij} = \texttt{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k=1}^{N} \exp(e_{ik})}. \tag{3.11}$$

Finally, the updated tokens $(\boldsymbol{x}'_i)_{i=1}^N$ are computed as a sum of the values weighted by the attention scores. For each $i \in \{1, \dots, N\}$,

$$\boldsymbol{x}'_i = \sum_{j=1}^{N} a_{ij} \boldsymbol{v}_j. \tag{3.12}$$

In matrix form, the full single-head self-attention mechanism can be written as follows, where $\boldsymbol{X}$ is the matrix whose rows are the tokens $\boldsymbol{X}_i = \boldsymbol{x}_i$.

$$\text{SHSA}(\boldsymbol{X}) = \texttt{softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_Q(\boldsymbol{X}\boldsymbol{W}_K)^\top}{\sqrt{d_{kq}}}\right)\boldsymbol{X}\boldsymbol{W}_V, \tag{3.13}$$

where the softmax function is applied row-wise.

3.5.2 Multi-head Self-Attention

It is empirically observed that extending the single-head self-attention mechanism to a multi-head self-attention (MHSA) mechanism improves performance [17]. The general belief is that this is caused because the multiple heads allow the model to jointly attend to information from different representation subspaces at different positions [17].

The MHSA mechanism operates by computing H different instances of single-head self-attention – each with their own learnable $\boldsymbol{W}_Q^i, \boldsymbol{W}_K^i, \boldsymbol{W}_V^i$ – and concatenating the results. Each of the results is then linearly projected using $\boldsymbol{W}_O^i \in \mathbb{R}^{d_{val} \times d}$ and added together to form the final output. In formulae, this can be written as

$$\texttt{MHSA}(\boldsymbol{X}) = \sum_{i=1}^{H} \texttt{SHSA}^i(\boldsymbol{X}) \boldsymbol{W}_O^i \tag{3.14}$$

$$= \sum_{i=1}^{H} \texttt{softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_Q^i (\boldsymbol{X}\boldsymbol{W}_K^i)^\top}{\sqrt{d_{kq}}}\right) \boldsymbol{X}\boldsymbol{W}_V^i \boldsymbol{W}_O^i : \tag{3.15}$$

Note that this definition is equivalent to the one given in the original Transformer paper [17].

3.6 Positional and Structural Encodings

The self-attention mechanism described above is agnostic to the input graph that connects the nodes. To give the model access to this information, positional and/or structural encodings can be added or appended to the node feature vectors. Positional encodings (PE) are meant to provide an idea of the *position in space* of a given node within the graph, while structural encodings (SE) are meant to provide an embedding of the *structure of the graph* itself. This manifests as follows: when two nodes are close to each other within a graph or subgraph, their PE should be similar; when two nodes share similar subgraphs, their SE should be similar [67]. Positional encodings and structural encodings can be applied to the node features or to the edge features.

In this dissertation, we formalize a positional or a structural encoding as a function $\nu : \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times d_{PE}}$ that enhances the node features, or a function $\mu : \mathbb{R}^{N \times N} \to \mathbb{R}^{M \times d_{PE}}$ that enhances the edge features. These functions take the adjacency matrix $\boldsymbol{A}$ as input and return a matrix of enhancements that are applied to the node or edge features through adding or concatenation. In formulae, this can be written as

$$\boldsymbol{X}' = \boldsymbol{X} + \nu(\boldsymbol{A}), \tag{3.16}$$

$$\boldsymbol{E}' = \boldsymbol{E} + \mu(\boldsymbol{A}), \tag{3.17}$$

or

$$\boldsymbol{X}' = \begin{bmatrix} \boldsymbol{X} & \nu(\boldsymbol{A}) \end{bmatrix}, \tag{3.18}$$

$$\boldsymbol{E}' = \begin{bmatrix} \boldsymbol{E} & \mu(\boldsymbol{A}) \end{bmatrix}. \tag{3.19}$$

In this dissertation, we will assume that the encodings are concatenated to the node and edge features.

Note that these encodings are fundamentally different from the positional encodings used in Transformers for sequence processing (e.g. in natural language processing). There, the positional encodings are used to give the model access to the order of the tokens (e.g. the sinusoidal encodings from [17]). In contrast, in graph machine learning, the positional encodings are used to give the model access to the position of the nodes in the graph, while the mechanism is kept equivariant w.r.t. node index permutations.

Fortunately, positional and structural encodings are well studied in the graph machine learning literature, and there is a variety of encodings to choose from.

An example of a positional encoding on nodes that has been shown to be effective and that we will use in our experiments consists of the eigenvectors of the Laplacian of the adjacency matrix [86]. The (symmetric normalized) Laplacian [87] of the adjacency matrix is defined as

$$\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}, \tag{3.20}$$

where $\boldsymbol{D}$ is the degree matrix of $\boldsymbol{A}$. The eigenvectors of this matrix can be used as a basis for the positional encodings, as they encode the structure of the graph in a way that is equivariant to node index permutations. If $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ is the eigendecomposition of $\boldsymbol{L}$ (which is guaranteed to exist, as L is symmetric if the graph is undirected), then the *Laplacian positional encoding* [86] can be defined as

$$\nu(\boldsymbol{A}) = \boldsymbol{U} \in \mathbb{R}^{N \times N} \tag{3.21}$$

If desired, only the first d_{PE} eigenvectors can be used by truncating $\boldsymbol{U}$, or the eigenvalues can be used along with the eigenvectors to provide additional information.

An example of a structural encoding on edges could be simply the weight of the edge. In this case, $\mu(\boldsymbol{A}) \in \mathbb{R}^M$ where

$$\mu(\boldsymbol{A})_i = \boldsymbol{A}_{\sigma(i),\tau(i)}, \tag{3.22}$$

and σ, τ are the functions that map the edge index to the source and target node indices, respectively.

Other commonly used examples of positional encodings include SignNet [78], identifiers for each connected component of a graph, and pair-wise node distances [79]. Some examples of structural encodings include the degree of a node [37], the diagonal of the m-step random-walk matrix [77] and the Ricci curvature at a node [36]. For a more in-depth overview of positional and structural encodings, we refer the reader to [67].

3.7 GraphGPS framework

GraphGPS is a framework for building graph Transformers, introduced in [67] and later employed to create the Exphormer method [21]. The framework consists of many sequentially stacked layers of the type depicted in Figure 3.1. The l'th layer (starting indices at 0) takes in node and edge feature matrices $\boldsymbol{X}^{(l)}$ and $\boldsymbol{E}^{(l)}$ and transforms them into new node and edge feature matrices $\boldsymbol{X}^{(l+1)}$ and $\boldsymbol{E}^{(l+1)}$. Each

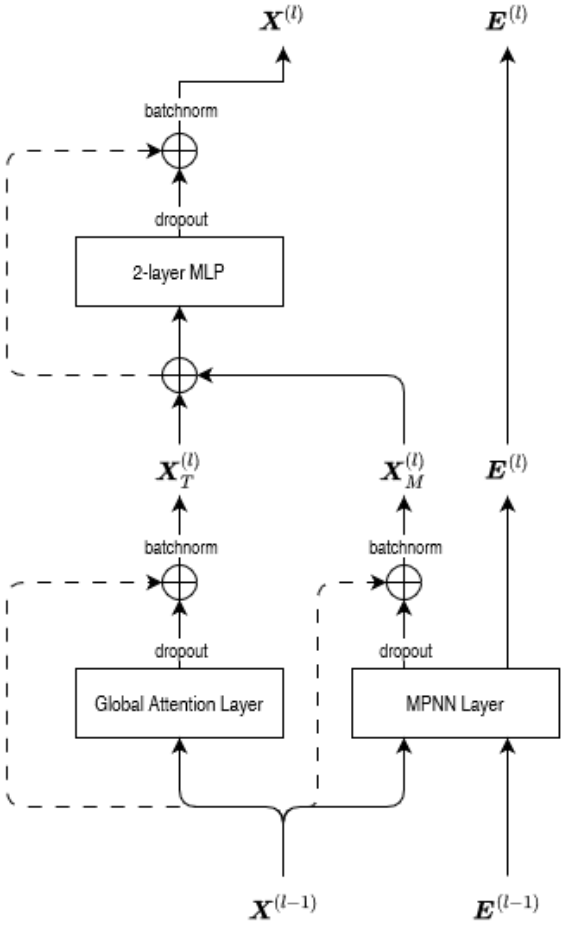


Figure 3.1: One layer in the GraphGPS framework. The dashed lines indicate residual connections.

layer consists of three main components: the *MPNN Layer*, the *Global Attention Layer* and the *2-layer MLP*.

The MPNN Layer is a traditional message-passing neural network layer that updates the node and edge features by aggregating information from neighbouring nodes and edges in the given graph, as explained in section 3.3. The MPNN Layers that we include are GCN [33], GAT [34], GINE (Graph Isomorphism Network with Edges) [84] and GatedGCN [85]. We employ GatedGCN for most experiments, as it has been observed in the GraphGPS paper to lead to the best performance [67].

While the MPNN Layer is limited to message-passing along the given graph, the Global Attention Layer ensures that nodes that are not directly connected can attend to each other. The Global Attention Layers that we include are Transformer (i.e.

full attention) [17], BigBird [52], Performer [46], Exphormer [21] and our own k-MIP attention mechanism. All of these, except from Exphormer, allow *any two nodes* to attend to each other.

The 2-layer MLP is a simple two-layer feedforward neural network that is applied to the node after the MPNN and Global Attention Layers.

A couple of elaborations are in order:

- A separate dropout layer can be applied within the Global Attention Layer, that randomly drops out a fraction of the attention scores. This is believed to improve generalisation [88].
- Of the mentioned MPNN Layers, only GatedGCN updates the edge attributes. We do not include a residual connection from $\boldsymbol{E}^{(l-1)}$ to $\boldsymbol{E}^{(l)}$, as GatedGCN already has an internal residual connection.
- Similarly, no residual connection is applied from $\boldsymbol{X}^{(l-1)}$ to $\boldsymbol{X}_M^{(l)}$ if the MPNN Layer is a GatedGCN.

3.8 Node Colourings and the SEG-WL Test

As discussed in section 2.4, node colouring algorithms like the Weisfeiler-Lehman test have been used to show important upper and lower bounds on the expressive power of graph neural networks. In section 5.1, we will contribute to this literature by show a similar upper bound on the expressive power of the GraphGPS framework, by employing the SEG-WL test from [65]. This section provides an overview of the necessary background to understand node colouring algorithms and the SEG-WL test.

3.8.1 Node Colourings and the 1-WL test

Notation 3. *We denote a* multiset *of elements by* $\{\!\{a_1, \ldots, a_n\}\!\}$*, where the order of the elements does not matter. We denote the set of class of multisets with elements from a class* $\mathcal{S}$ *by* $\mathbb{N}^{\mathcal{S}}$*.*

Definition 4. *A* node colouring *of a graph G is a function $c : V \to \mathcal{C}$ that assigns a colour to each node in G. Here, $\mathcal{C}$ denotes the set of possible colours.*

Note that there is no restriction on the class $\mathcal{C}$. In particular, $\mathcal{C}$ could be $\mathbb{R}^d$, therefore any GNN and every instance of the GraphGPS framework can be seen as an algorithm that generates node colourings.

The Weisfeiler-Lehman test (abbreviated 1-WL test, to distinguish it from higher-order variations [13]) is such a node colouring algorithm that was originally introduced to test graph isomorphism. The test iteratively refines the node colouring of a graph by aggregating the colours of neighbouring nodes, as described in the following definition and illustrated in Figure 3.2.

Definition 5. *The* Weisfeiler-Lehman Test *(1-WL Test) is a graph isomorphism test that iteratively refines the node colouring of a graph as described below.*

Let the input be a graph $G = (V, E)$ with initial node colouring $c^{(0)} : V \to \mathcal{C}$. The test iteratively updates the colour of each node $v \in V$ as

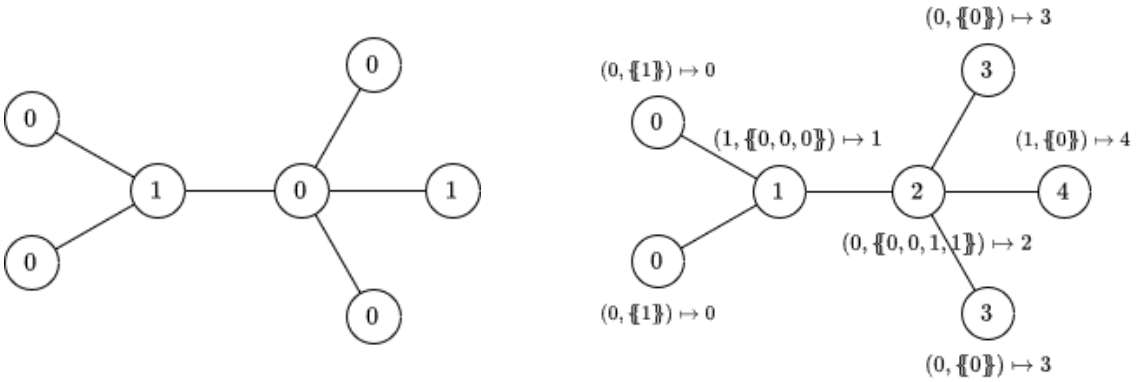
$$c^{(l)}(v) = \tau\left(c^{(l-1)}(v), \left\{\!\!\left\{ c^{(l-1)}(r) \mid r \in \mathcal{N}_{in}(v) \right\}\!\!\right\}\right), \tag{3.23}$$

where τ is an injective map from $\mathcal{C} \times \mathbb{N}^{\mathcal{C}}$ to $\mathcal{C}$.

The algorithm can be terminated after a fixed number of iterations L, or when the equivalence classes of the node colourings stabilise.

The 1-WL test can be used to test whether two graphs are isomorphic, by comparing the multisets of node colours generated by the test on the two graphs. If the multisets are different, the graphs are guaranteed to be non-isomorphic. In this case, the 1-WL test is said to distinguish both graphs. However, if the multisets are the same, they may or may not be isomorphic. This idea can be extended to other node colouring algorithms as follows.

Definition 6. *An algorithm $\mathcal{A}$ that generates node colourings* distinguishes *two non-isomorphic graphs G_1 and G_2 iff the multisets of node colourings generated by*



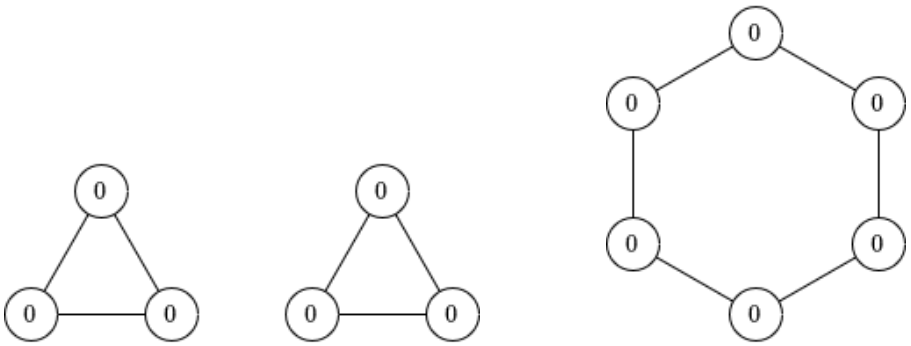
(a) Initial node colouring (iteration 0) (b) Node colouring after 1 iteration

Figure 3.2: Illustration of a single iteration of the Weisfeiler-Lehman test. The node labels (in $\mathcal{C} = \mathbb{N}$) are written inside the nodes. The mapping τ is written next to the nodes.

$\mathcal{A}$ on G_1 and G_2 are not equal, i.e. iff

$$\{\!\{\mathcal{A}(G_1)(v) \mid v \in V_1\}\!\} \neq \{\!\{\mathcal{A}(G_2)(v) \mid v \in V_2\}\!\}\,. \tag{3.24}$$

While the 1-WL test is a powerful tool for distinguishing pairs of non-isomorphic graphs, it cannot distinguish all such pairs. Figure 3.3 shows an example of two non-isomorphic graphs that are indistinguishable by the 1-WL test [89].



(a) Two disjoint 3-cycles (b) 6-cycle

Figure 3.3: Two graphs that are indistinguishable by the 1-WL test.

3.8.2 SEG-WL Test

As discussed in section 2.4, augmenting the input features with positional encodings can increase the power of graph neural networks for distinguishing non-isomorphic

graphs. In [65], the authors introduce the Structural Encoding Enhanced Global Weisfeiler-Lehman Test (SEG-WL test), which is a generalisation of the 1-WL test that incorporates structural encodings into the node colouring algorithm. Because we will characterise the expressive power of the GraphGPS framework – which includes the k-MIP-GT – using the SEG-WL test, we will define the SEG-WL test in the remainder of this section.

The following definition forms a general framework for feature augmentation, that can augment both node-wise and edge-wise features.

Definition 7 ([65])**.** *A* structural encoding scheme $S = (f_A, f_R)$ *is a pair of functions, where for any graph* $G = (V, E)$, $f_A(v, G) \in \mathcal{C}$ *defines the encoding of any node* $v \in V$ *and* $f_R(v, u, G) \in \mathcal{C}$ *defines the encoding of any node pair* $(v, u) \in V \times V$. S *is called* strongly regular *if there exists a discriminator function* $\xi : \mathcal{C} \times \mathcal{G} \to \{0, 1\}$ *such that* $\xi(f_R(v, u, G), G) = 1$ *if and only if* $u = v$.

For each structural encoding scheme S, there is an associated S-SEG-WL test, which is defined as follows.

Definition 8 ([65])**.** *The* S-SEG-WL Test*, where* $S = (f_A, f_R)$ *is a structural encoding scheme, is a graph isomorphism test that iteratively refines the node colouring of a graph as described below.*

Let the input be a graph $G = (V, E, \boldsymbol{X}, \boldsymbol{E}, \boldsymbol{A})$. *The test proceeds as follows:*

1. *Initialize the node colouring* $c^{(0)} : V \to \mathcal{C}$ *as*

$$c^{(0)}(v) = \Phi_0(\boldsymbol{X}_v, f_A(v, G)), \tag{3.25}$$

   where Φ_0 *is an injective function from* $\mathbb{R}^d \times \mathcal{C}$ *to* $\mathcal{C}$.

2. *In iteration* l*, compute the colour* $c^{(l)}(v)$ *of each node* $v \in V$ *as*

$$c^{(l)}(v) = \Phi(\{\!\{ \left(c^{(l-1)}(r), f_R(v, r, G)\right) \mid r \in V \}\!\}), \tag{3.26}$$

   where Φ *is a function that injectively maps* $\mathbb{N}^{\mathcal{C} \times \mathcal{C}}$ *to* $\mathcal{C}$.

3.8.3 The SEG-WL Preorder

Different node colouring algorithms can be compared in terms of their expressive power by comparing the pairs of graphs that they can distinguish. If an algorithm $\mathcal{A}$ can distinguish every pair of non-isomorphic graphs that $\mathcal{B}$ can distinguish, we say that $\mathcal{A}$ is *more expressive* than $\mathcal{B}$. If, in addition, there exists a pair of graphs that $\mathcal{A}$ can distinguish but $\mathcal{B}$ cannot, we say that $\mathcal{A}$ is *strictly more expressive* than $\mathcal{B}$. It is easy to check that the relation "is more expressive than" is reflexive and transitive, and therefore forms a preorder on the class of node colouring algorithms. Note that it is not antisymmetric, so it is not a partial order as was wrongly stated in [65].

As there is a SEG-WL test corresponding to each structural encoding scheme, the relation "is more expressive than" also forms a preorder on the class of SEG-WL tests. One result that provides insight in this preorder is Theorem 3 from [65], which embodies the intuitive result that SEG-WL tests become more expressive as their structural encoding schemes become more discriminative. To formalize this, we first introduce the notion of a *refinement* of a structural encoding scheme.

Definition 9. *Consider two structural encoding schemes $S = (f_A, f_R)$ and $S' = (f'_A, f'_R)$. We call S' a* refinement *of S (notated $S' \gtrsim S$) if there exist mappings p_A, p_R such that for any G and $u, v \in V$, we have*

$$f_A(v, G) = p_A(f'_A(v, G)), \tag{3.27}$$

$$f_R(v, u, G) = p_R(f'_R(v, u, G)). \tag{3.28}$$

Then the S-SEG-WL test is more expressive than the S'-SEG-WL test in testing non-isomorphic graphs.

Theorem 10 (Theorem 3 in [65])**.** *For two structural encoding schemes S and S', if $S' \gtrsim S$, then the S'-SEG-WL test is more expressive than the S-SEG-WL test. Further, if S-SEG-WL distinguishes two non-isomorphic graphs G_1 and G_2 after t iterations, then S'-SEG-WL distinguishes G_1 and G_2 after at most t iterations.*

4 | Method

In this chapter, we present our proposed k-MIP Graph Transformer (k-MIP-GT) architecture, which leverages the k-Maximum Inner Product (k-MIP) self-attention mechanism to improve the efficiency and scalability of graph machine learning models.

We first introduce the k-MIP self-attention mechanism (section 4.1), which is a core component of our method. An essential part of this mechanism is the k-Maximum Inner Product Search (k-MIPS) operation, which identifies the k keys with the highest inner products with a given query. In section 4.2, we give three methods for performing this search: naive brute-force, brute-force with symbolic matrices, and Ball Tree Search. The efficiency of all three approaches will be compared later in this dissertation, in sections 6.2 and 6.3.

Next, we describe how all components come together in the architecture of our k-MIP-GT model (section 4.3). Finally, we discuss the tools and technologies chosen for the implementation of our method (section 4.4), explaining their roles in achieving efficiency and scalability.

4.1 k-MIP Self-Attention

In the computation of the standard single-head self-attention mechanism (see subsection 3.5.1), every query $\boldsymbol{q}_i$ attends to every key $\boldsymbol{k}_j$ to compute a full attention score matrix $\boldsymbol{A} \in [0,1]^{N \times N}$ as

$$\boldsymbol{A} = \texttt{softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_Q(\boldsymbol{X}\boldsymbol{W}_K)^\top}{\sqrt{d_{kq}}}\right). \tag{4.1}$$

This operation has a quadratic time complexity, and requires a computation of N expensive softmax computations over N elements each. Further, in large graphs, one can expect that a lot of the keys are irrelevant to a given query, which is unnecessarily

inefficient and introduces noise in the process [90, 91].

To address these issues, we propose the following mechanism: for every query $\boldsymbol{q}_i$, we select the k keys K_i that have the highest inner product with $\boldsymbol{q}_i$, and only attend to those keys. The intuition behind this is that the keys in K_i are the keys with the largest attention scores, which suggests that they are the most relevant keys for the query $\boldsymbol{q}_i$. This mechanism has been used before in the context of natural language processing [26] and computer vision [25], but has not yet been named nor been used in the context of graph machine learning. Therefore, we name it *k-Maximum Inner Product (k-MIP) Self-Attention*. The mechanism is defined as follows:

$$e_{ij} = \frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d_{kq}}}, \tag{4.2}$$

$$K_i = \texttt{argtopk}_j(e_{ij}), \tag{4.3}$$

$$a_{ij} = \begin{cases} \texttt{softmax}_{j \in K_i}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{j' \in K_i} \exp(e_{ij'})} & \text{if } j \in K_i \\ 0 & \text{otherwise} \end{cases}, \tag{4.4}$$

$$\boldsymbol{x}'_i = \sum_{j=1}^{N} a_{ij} \boldsymbol{v}_j. \tag{4.5}$$

Note that the softmax is only performed on the k elements in K_i, such that $(a_{ij})_{j=1}^{N}$ still sums to 1 for every i.

In matrix form, the single-head k-MIP attention mechanism can be written as follows, where $\boldsymbol{X}$ is the matrix whose rows are the tokens $\boldsymbol{X}_i = \boldsymbol{x}_i$.

$$\texttt{k-MIP-Attn-SH}(\boldsymbol{X}) = \texttt{top-k-softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_Q(\boldsymbol{X}\boldsymbol{W}_K)^\top}{\sqrt{d_{kq}}}\right)\boldsymbol{X}\boldsymbol{W}_V. \tag{4.6}$$

Here, `top-k-softmax` first selects the k highest values per row and then performs a row-wise softmax operation on the selected values.

$$\texttt{top-k-softmax}(\boldsymbol{A})_{ij} = \begin{cases} \frac{\exp(\boldsymbol{A}_{ij})}{\sum_{j' \in \text{top}_k(\boldsymbol{A}_{i\cdot})} \exp(\boldsymbol{A}_{ij'})} & \text{if } j \in \text{top}_k(\boldsymbol{A}_{i\cdot}) \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

Even though the time complexity of this operation remains $\mathcal{O}(N^2)$, several computational advantages make this approach more efficient and scalable compared to full self-attention, which we will empirically demonstrate in section 6.2:

- The softmax operation is performed on k elements per row (instead of N), and only k weighted values need to be aggregated per row (instead of N). These two effects combine to a 5x speedup during inference.
- During training, the number of intermediate inner product results that need to be stored is reduced to Nk (instead of N^2), and similarly, only Nk gradients for these intermediate inner product results need to be computed during the backward pass (instead of N^2). This leads to a 10x speedup during training (see section 6.2), as well as a negligible memory footprint.
- The inherent sparsity enables the use of symbolic matrices for the k-MIPS operation, further accelerating both training and inference by another order of magnitude.

In addition to these computational gains, the k-MIP self-attention mechanism offers two further benefits:

- The hyperparameter k allows us to control the trade-off between computational efficiency and model performance (see section 6.4).
- The attention mechanism inherently focuses on the keys that have the largest impact on the output, potentially making it more robust to noise by filtering out less important or noisy information.

This mechanism is extended to multi-head k-MIP self-attention in the same way as standard multi-head self-attention (see section 3.5). This leads to the definition

$$\texttt{k-MIP-Attn}(\boldsymbol{X}) = \sum_{i=1}^{H} \texttt{top-k-softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_Q^i(\boldsymbol{X}\boldsymbol{W}_K^i)^\top}{\sqrt{d_{kq}}}\right)\boldsymbol{X}\boldsymbol{W}_V^i\boldsymbol{W}_O^i. \tag{4.8}$$

4.2 k-Maximum Inner Product Search

An important component of the k-MIP self-attention mechanism – and its bottleneck when the graph is large – is the computation of $K_i = \texttt{argtopk}_j(e_{ij})$ for each i as in (4.3). For each query $\boldsymbol{q}_i$, this operation requires finding the indices K_i of the k keys $\{\boldsymbol{k}_j \mid j \in K_i\}$ that have the highest inner products with $\boldsymbol{q}_i$. In matrix form, this operation can be formulated as finding

$$\texttt{argtopk}(\boldsymbol{Q}\boldsymbol{K}^\top) \in \mathbb{R}^{N \times k}, \tag{4.9}$$

where the `argtopk` operation is row-wise and $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K$ are matrices whose rows are the queries and keys, respectively.

This problem of k-Maximum Inner Product Search (k-MIPS) is relatively well-studied in the Information Retrieval literature. However, due to the lack of surveys and benchmarks that compare the efficiency of different approaches, as well as the fact that there is no one-size-fits-all solution, we opted to adopt various approaches and assess their performance empirically.

In this work, we examine three primary approaches for exact k-MIPS: naive brute-force search, brute-force search with symbolic matrices, and Ball Tree Search. Although we experimented with various open-source libraries for approximate k-MIPS as well, including FAISS [92], Spotify Annoy [93], and TorchPQ [94], our preliminary experiments revealed that the efficiency gains they offered were not sufficient to justify the loss of exact results. Consequently, we excluded these methods from further consideration.

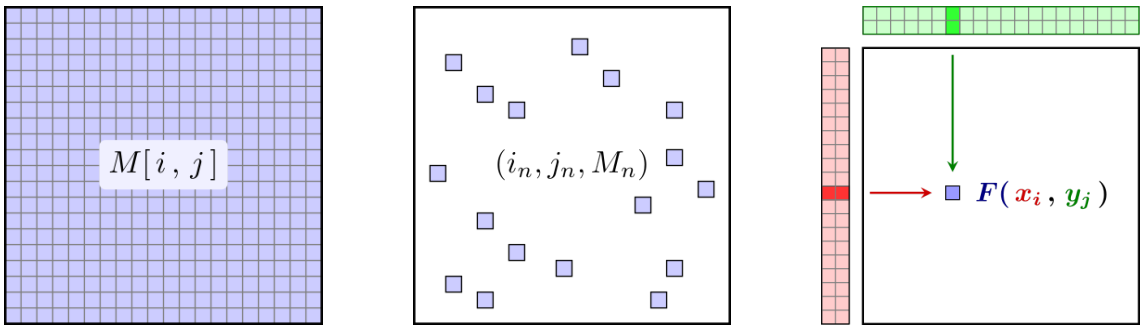
We present a comparative analysis of naive brute-force search and brute-force search with symbolic matrices in section 6.2, while we provide our results using Ball Tree Search in section 6.3. We will now describe each of those three approaches in detail.

4.2.1 Naive Brute-Force Search

The naive brute-force search first computes for each query $\boldsymbol{q}_i$ the row vector $\boldsymbol{q}_i \cdot \boldsymbol{K}^\top$, which contains its inner product with every key $\boldsymbol{k}_j$. This is usually done in parallel for all queries (or per batch of queries) using matrix multiplication. Then, for each such row vector, the k largest elements are found using a partial sort operation. This operation requires the computation of $\mathcal{O}(N^2)$ inner products of dimension d_{kq}, and therefore has a time complexity that is quadratic in the number of nodes N.

4.2.2 Brute-Force Search with Symbolic Matrices

While the naive implementation of brute-force k-MIPS is straightforward, it is often prohibitively slow due to the need to compute and store N^2 inner products. However, one can significantly improve both the time and memory efficiency of this approach by making better use of GPU acceleration. To this end, we propose to use symbolic matrices as provided by the PyKeOps library [95].



(a) Dense Matrix (b) Sparse Matrix (c) Symbolic Matrix

Figure 4.1: A comparison of dense, sparse, and symbolic matrices. Adopted from [95].

Three ways to store matrices are depicted in Figure 4.1: dense, sparse, and symbolic matrices. Typical machine learning frameworks use dense matrices to store data (Figure 4.1a), which store each element in memory and can thus have a large memory footprint and slow storage and retrieval times. For specific operations, sparse matrices (Figure 4.1b) can be used to store only the indices and values of a small number of non-zero elements, which can significantly reduce the memory footprint and computation time. Symbolic matrices (Figure 4.1c) are a third way

to store data, whose coefficients are given by a formula $\boldsymbol{M}_{i,j} = F(\boldsymbol{x}_i, \boldsymbol{y}_j)$ that is evaluated on data arrays $(\boldsymbol{x}_i)_{i=1}^{N}$ and $(\boldsymbol{y}_j)_{j=1}^{M}$. Reduction operations are evaluated lazily, with high levels of parallelism, and computed without ever sending intermediate results to the GPU's global memory, which can make them 30 to 1000 more efficient than their dense counterparts [95].

In our case, we use symbolic matrices to compute, for all queries $\boldsymbol{q}_i$, the k keys $\boldsymbol{k}_j$ that have the highest inner products with $\boldsymbol{q}_i$. We first define the symbolic matrix $\boldsymbol{E}$ with the formula $\boldsymbol{E}_{i,j} = \boldsymbol{q}_i \cdot \boldsymbol{k}_j$. Then, we apply a reduction on the rows of $\boldsymbol{E}$ that computes the indices of the k largest elements of each row.

Because of our use of symbolic matrices, this computation never materialises the results of the N^2 inner products in main GPU memory and instead makes use of the GPU's shared memory and registers. The result is that our implementation of k-MIP self-attention has a negligible memory footprint for both the forward pass and backpropagation and is two orders of magnitude faster than the PyTorch implementation of the full self-attention mechanism. We will demonstrate the latter later in this dissertation, in section 6.2.

4.2.3 Ball Tree Search

While brute-force k-MIP search with symbolic matrices is significantly more efficient than the naive brute-force search, it still has a quadratic time complexity due to the need to compute N^2 inner products.

Aiming to reduce this complexity and inspired by the speedups achieved by [96], we adopted their Algorithm 5 to perform exact k-MIP search using a Ball Tree Search. The algorithm proceeds in two steps: first, it constructs a ball tree data structure to store the keys, and then it searches this ball tree once for each query to find the k keys with the largest inner product with that query. The idea is that the construction step takes $\mathcal{O}(N \log N)$ and each search takes $\mathcal{O}(\log N)$, resulting in a total complexity of $\mathcal{O}(N \log N)$. We will now describe this algorithm in more detail.

The ball tree data structure is a binary tree where each (tree) node represents a

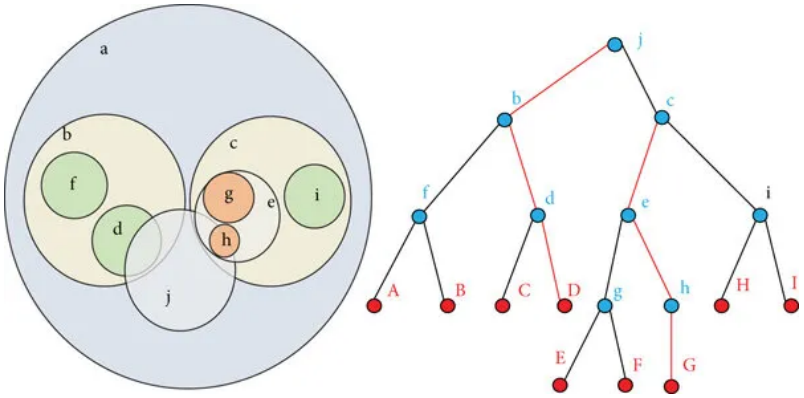


Figure 4.2: Diagram illustrating a ball tree data structure. Adopted from [97].

cluster of key vectors encapsulated within a hyperball, defined by a centre point and a radius, that contains all keys in that node's subtree. See Figure 4.2 for an example of such a data structure. To construct the ball tree, the algorithm recursively partitions the keys into two clusters, and then computes the smallest hyperball centred at the centroid of each cluster that contains all keys in that cluster. This process is repeated until each leaf node contains at most `maxLeafSize` keys, at which point the algorithm has constructed the ball tree.

To search the ball tree for a given $\boldsymbol{q}_i$ (i.e. find the k keys with the highest inner product with $\boldsymbol{q}_i$), the algorithm employs a branch-and-bound strategy while traversing the tree in a depth-first manner. Specifically, it maintains a priority queue of the k highest inner products found so far. As it traverses the tree, it computes an upper bound for each visited hyperball, representing the highest possible inner product between $\boldsymbol{q}_i$ and any key within that hyperball. If this upper bound is lower than the lowest value in the priority queue, the algorithm skips the subtree rooted at that node. Additionally, the search is guided to first visit the (tree) node whose hyperball has the highest upper bound, ensuring that the algorithm explores the most promising branches first and prunes the search space more effectively.

To bound the highest possible inner product between a given query $\boldsymbol{q}$ and any vector within a hyperball B with centre $\boldsymbol{c}$ and radius r, the algorithm uses the following bound proven in [96]:

$$\max_{\boldsymbol{x} \in B} \boldsymbol{q}^T \boldsymbol{x} \leq \boldsymbol{q}^T \boldsymbol{c} + r \left\| \boldsymbol{q} \right\| \tag{4.10}$$

The idea of this Ball Tree Search algorithm is that the ball tree construction has a complexity of $\mathcal{O}(N \log N)$, and each search has a complexity of $\mathcal{O}(\log N)$, which results in a total complexity of $\mathcal{O}(N \log N)$. This would be significantly better than the naive brute-force approach, which has a complexity of $\mathcal{O}(N^2)$. While we will demonstrate that the algorithm indeed has a complexity of $\mathcal{O}(N \log N)$ in section 6.3, we will also show that the algorithm's efficiency is not competitive with the symbolic matrix brute-force search in practice.

4.3 Architecture

To build a graph Transformer, we employ the GraphGPS framework [67] that we described in section 3.7 and use the k-MIP self-attention mechanism as the Global Attention Layer. We will now describe the full architecture of our method.

Let $G = (V, E, \boldsymbol{X}, \boldsymbol{E})$ be the input graph and let $\boldsymbol{A}$ be its weighted adjacency matrix. First, the node and edge features are enhanced with positional and structural encodings, as described in section 3.6. Concretely, the enhanced node feature and edge feature matrices are obtained as

$$\boldsymbol{X}^{(0)} = \begin{bmatrix} \boldsymbol{X} & \nu(\boldsymbol{A}) \end{bmatrix}, \tag{4.11}$$

$$\boldsymbol{E}^{(0)} = \begin{bmatrix} \boldsymbol{E} & \mu(\boldsymbol{A}) \end{bmatrix}, \tag{4.12}$$

where $\nu(\boldsymbol{A}) \in \mathbb{R}^{N \times d_{PE}}$ and $\mu(\boldsymbol{A}) \in \mathbb{R}^{M \times d_{PE}}$ are the node and edge enhancements, respectively.

Next, the node and edge features are passed through a series of L GraphGPS layers, each consisting of a Global Attention Layer, a MPNN Layer and a 2-Layer MLP. For the Global Attention Layer, the k-MIP self-attention mechanism is used. These layers iteratively compute node and edge feature matrices $(\boldsymbol{X}^{(l)})_{l=1}^{L}$ and $(\boldsymbol{E}^{(l)})_{l=1}^{L}$ as

follows:

$$\boldsymbol{X}_M^{*(l)}, \boldsymbol{E}^{(l)} = \texttt{MPNN}(\boldsymbol{X}^{(l-1)}, \boldsymbol{E}^{(l-1)}), \tag{4.13}$$

$$\boldsymbol{X}_M^{(l)} = \texttt{BatchNorm}\big(\texttt{Dropout}(\boldsymbol{X}_M^{*(l)}) + \boldsymbol{X}^{(l-1)}\big), \tag{4.14}$$

$$\boldsymbol{X}_T^{(l)} = \texttt{BatchNorm}\big(\texttt{Dropout}(\texttt{k-MIP-Attn}(\boldsymbol{X}^{(l-1)})) + \boldsymbol{X}^{(l-1)}\big), \tag{4.15}$$

$$\boldsymbol{X}^{(l)} = \texttt{BatchNorm}\big(\texttt{Dropout}(\texttt{MLP}(\boldsymbol{X}_M^{(l)} + \boldsymbol{X}_T^{(l)})) + \boldsymbol{X}_M^{(l)} + \boldsymbol{X}_T^{(l)}\big). \tag{4.16}$$

Such a single layer of the k-MIP-GT is depicted in Figure 4.3.

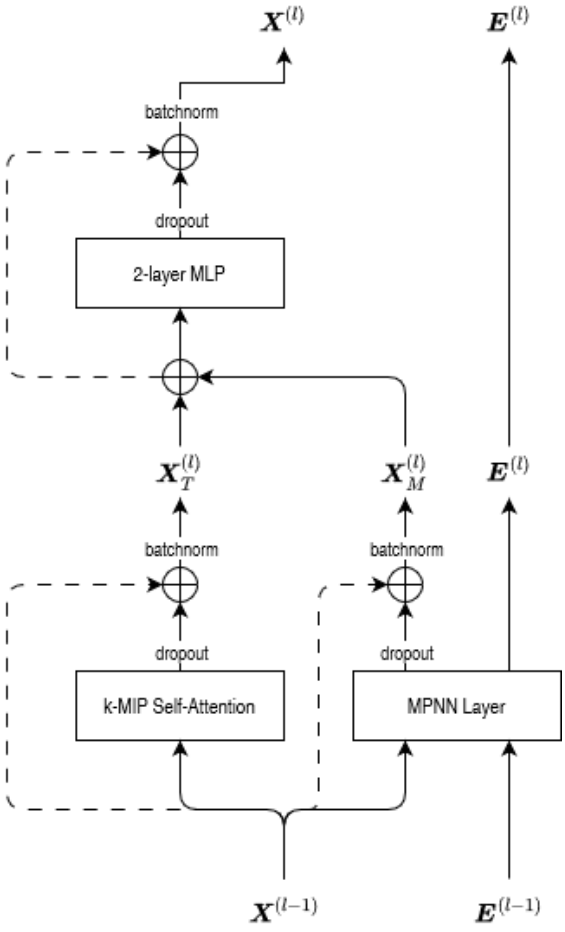


Figure 4.3: One layer in the k-MIP-GT. The dashed lines indicate residual connections.

The final step depends on the task that is being solved. For node-level tasks, the node features $\boldsymbol{X}^{(L)}$ are returned as-is. For graph-level tasks, the node features are aggregated into a single graph feature vector $\boldsymbol{x}_G$ using a permutation invariant

global pooling function, such as sum or mean aggregation. For node-pair-level tasks (which includes edge-level tasks), a feature $\boldsymbol{x}_{ij}$ is decoded per node pair based on the features of both nodes in the pair.

This architecture is designed to be extremely versatile. It supports edge feature, can be used for a wide range of tasks (including node-level, edge-level, and graph-level tasks) and supports a variety of MPNNs and positional encodings.

4.4 Choice of Technology

Software engineering made up a significant portion of this project, as we had to implement our proposed method, experimental setups, and data-loading pipelines. To achieve this efficiently, we leveraged several powerful tools and technologies, each chosen for its specific advantages:

Python [98] served as the primary programming language throughout the project, providing flexibility and extensive libraries for machine learning and scientific computing.

PyTorch [99] was the deep learning framework used to build, train, and evaluate our models. Its automatic differentiation capabilities, support for GPU acceleration, and intuitive API made it an ideal choice for implementing our method. Additionally, we developed a custom PyTorch C++ extension to optimise the Ball Tree Search for k-MIPS.

PyKeOps [95] was employed for implementing the k-MIPS operation, taking advantage of symbolic matrices to accelerate computations and manage GPU memory efficiently.

PyTorch Geometric [100] was used to handle graph data structures and implement our method. We also used PyTorch Geometric for loading and preprocessing the datasets.

GraphGym [101] offered a modular and extensible framework for designing and evaluating graph machine learning models. We integrated our method into GraphGym as a custom 'network,' ensuring flexibility in experimentation.

GraphGPS [67] Since our method builds on the GraphGPS framework, we utilised its existing codebase as a starting point for implementation. We made significant modifications to tailor it to our needs, including support for new datasets and the implementation of our k-MIP-GT. These contributions will be shared back with the GraphGPS repository after the marking period.

Weights & Biases [102] was used for experiment tracking and logging. It allowed us to efficiently monitor the performance of our models throughout the training process.

Matplotlib [103] was used to generate the plots and visualisations included in this dissertation.

5 | Theoretical Results

The objective of this chapter is to characterise the expressive power of our proposed k-MIP Graph Transformer, which we do in the following three steps.

First, in section 5.1 we prove that any instance of the GraphGPS framework – which includes our k-MIP-GT, as well as Exphormer [21] and GPS++ [74] – can only distinguish graphs that are distinguishable by the SEG-WL test that uses the same positional and structural encodings. While this limitation can be overcome by using expressive encodings, this result highlights the importance of said encodings in the GraphGPS framework and puts expressivity claims of various previous works into a new perspective.

Second, we show in section 5.2 that a sequence Transformer equipped with k-MIP self-attention is a universal approximator for continuous sequence-to-sequence functions, just like the standard Transformer [27]. This is an important result, as it guarantees that sparsifying the self-attention mechanism does not come at the cost of expressiveness.

Finally, we extend this result to the graph setting in section 5.3. Under the same assumptions as [21] and [28], we show that the k-MIP-GT is a universal approximator for continuous functions on graphs. Again, this is essential, as it demonstrates that even in the graph setting, the same expressivity result can be shown despite the sparsification of the self-attention mechanism.

All proofs are provided in section 5.4.

To show these theoretical results, we slightly simplify the architecture presented in section 4.3 by omitting batch normalization. This simplification is necessary to make the output of the model only dependent on a single input graph, which enables the characterization of the model as a mapping. This mapping is also deterministic, because dropout is disabled at inference time. Further, we focus on node-level tasks, where no graph pooling or edge decoding is applied and instead the node features

$\boldsymbol{X}^{(L)}$ are directly returned without additional processing. The results we present are, however, easily extensible to graph- and edge-level tasks.

Further, we will use the following properties of the components of the GraphGPS framework. Note that the assumption of these properties is not restrictive, as they are satisfied by all the instances of `MPNN`, `Attn`, and `MLP` that are implemented in the GraphGPS framework. In particular, we do not assume yet that `Attn` is the k-MIP self-attention mechanism, as our result in section 5.1 holds for any instance of the GraphGPS framework.

Assumption 11. *We assume the following properties of `MPNN`, `Attn`, and `MLP`:*

A1. The output embedding of `MPNN` for node v only depends on the previous embedding of v, the embeddings of the nodes in the in-neighbourhood of v and the edge embeddings of the corresponding incoming edges. Further, it is invariant w.r.t. permutations of the neighbours.

A2. The output embedding of `MPNN` for edge (r, v) only depends on the previous embedding of (r, v) and the previous embeddings of the nodes r and v.

A3. `Attn` is equivariant w.r.t. token permutations, and thus to permutations of the node indices.

A4. `MLP` acts on each node embedding independently.

5.1 Upper Bound on Expressiveness

In this section, we prove that the GraphGPS framework – which includes k-MIP-GT, Exphormer [21], and GPS++ [74] – has an upper bound on its expressive power in terms of graph distinguishability. Concretely, we show that GraphGPS can only distinguish graphs that are also distinguishable by the SEG-WL test (introduced in section 3.8) enhanced with the same positional and structural encodings. This result allows us to compare the expressive power of GraphGPS to the 1-WL test and highlights the importance of expressive encodings in the framework. Further, we will use it to shed new perspective on the super-1-WL expressiveness results

given in various previous works [67, 28, 21].

Importantly, it should be noted that this result is not necessarily a drawback to our method, as we will show in section 5.3 that the k-MIP-GT can universally approximate functions on graphs when an expressive positional encoding is used.

5.1.1 Results

In the following theorem, we state and prove that the expressive power of the GraphGPS framework in terms of graph distinguishability is upper bounded by the analogous expressivity of the S-SEG-WL test that uses the same node and edge feature enhancements. Note that, by consequence, this also puts an upper bound on its expressive power for function approximation, as the model cannot represent any function that assigns different outputs to graphs indistinguishable by this S-SEG-WL test.

Theorem 12. *Let $\mathcal{A}$ be an instance of the GraphGPS framework that enhances its node features with $\nu(\boldsymbol{A})$ and its edge features with $\mu(\boldsymbol{A})$. Then $\mathcal{A}$ can only distinguish graphs that are also distinguishable by the S-SEG-WL test, where $S = (f_A, f_R)$ and f_A, f_R are defined as*

$$f_A(v, G) = \nu(\boldsymbol{A})_v, \tag{5.1}$$

$$f_R(v, u, G) = \begin{cases} (0, \mathbb{1}_{u=v}, \mathbf{0}_{d_{edge}}, \mathbf{0}_{d_{PE}}) & \text{if } (u, v) \notin E \\ (1, \mathbb{1}_{u=v}, \boldsymbol{E}_{uv}, \mu(\boldsymbol{A})_{uv}) & \text{if } (u, v) \in E \end{cases}, \tag{5.2}$$

where $\mathcal{C} = \mathbb{R}^{d_{PE}} \cup \{0, 1\}^2 \times \mathbb{R}^{d_{edge}} \times \mathbb{R}^{d_{PE}} \cup \mathbb{R}^d \cup \mathbb{R}^{d_{edge}}$.

The proof of this theorem can be found in subsection 5.4.1.

By establishing a connection between the expressive power of GraphGPS and the S-SEG-WL test in Theorem 12, we have made it possible to relate the expressive power of the GraphGPS framework (and hence, of our k-MIP-GT) to both the 1-WL test and to other graph Transformers. We will discuss three implications of this result. First, we compare the expressive power of GraphGPS to the 1-WL test.

Second, we investigate the effect of more expressive encodings on the expressive power of GraphGPS. Third, we discuss the origin of the expressive power of graph Transformers.

Comparison to the 1-WL Test First, we note that the S-SEG-WL test is a generalization of the 1-WL test, as the 1-WL test can be recovered by setting, for all $u, v \in V$,

$$f_A(v, G) = c_0, \tag{5.3}$$

$$f_R(v, u, G) = \begin{cases} c_1 & \text{if } (u, v) \in E \\ c_2 & \text{if } (u, v) \notin E \end{cases}, \tag{5.4}$$

where $c_0, c_1, c_2 \in \mathcal{C}$ are constants.

This has profound implications for GraphGPS. In particular, when considering the standard setting for evaluating 1-WL expressiveness where all node features are identical, all edge features are identical and no positional encodings are used, note that the structural encoding scheme S from Theorem 12 degenerates to the form (5.3)-(5.4). Hence, in that case, Theorem 12 implies that the expressive power of the GraphGPS framework in terms of graph distinguishability is upper bounded by that of the 1-WL test.

When More Expressive Encodings are Used As we formally introduced in subsection 3.8.3, there is a preorder on the expressive powers of S-SEG-WL tests. In particular, if S, S' are structural encoding schemes and S' is a refinement of S (see subsection 3.8.3), then S'-SEG-WL is at least as expressive in terms of graph distinguishability as S-SEG-WL. While this theorem does not state that S'-SEG-WL is strictly more expressive than S-SEG-WL, [65] provide various examples where the order relation is strict.

The implication for GraphGPS is that more expressive positional and structural encodings lead to a higher upper bound on the expressiveness of the framework. In particular, using the Laplacian positional encoding allows GraphGPS to distinguish

some graphs that are indistinguishable by the 1-WL test. An example of such a pair is given in [28]. Combining this with the fact that the MPNN module in the GraphGPS framework can be equally expressive as the 1-WL test when an expressive MPNN is used (e.g. GINE) [14], we can conclude that GraphGPS with Laplacian positional encoding is strictly more expressive than the 1-WL test.

The Origin of Graph Transformers' Expressive Power While various previous works on graph Transformers [67, 21, 28] have claimed super-1-WL expressiveness for their methods (in terms of graph distinguishing power), our result highlights that this expressiveness comes from the positional and structural encodings applied to the node and edge features, rather than from the Transformer architecture itself. In particular, for all of the three mentioned works, the same level of graph distinguishing power could be attained using an expressive GNN where the input features are augmented with the same positional and structural encodings.

5.1.2 Discussion

In Theorem 12, we demonstrated that the expressive power of the GraphGPS framework is fundamentally constrained by the positional and structural encodings applied to the node and edge features. We argued that, without enhancement of these features, the framework's expressivity is even upper bounded by the 1-WL test. This result underscores the importance of expressive encodings for making GraphGPS more powerful.

In our experiments in chapter 6, we primarily rely on the Laplacian positional encoding (see section 3.6), which is an example of an encoding that allows super-1-WL expressiveness [28].

The perspective offered by Theorem 12 highlights some nuances about claims in the literature. For instance, Exphormer [21] and SAN [28] ascribe their super-1-WL expressiveness to the Transformer's universal approximation properties. However, like our own method, a closer examination reveals that the same level of graph distinguishing power could be attained using an expressive GNN with identical features

augmentations. This follows from Theorem 12 for Exphormer and has been argued in [65] for SAN. Hence, the true source of this expressiveness lies in the positional encodings rather than the Transformer network itself. Acknowledging the role of these encodings is crucial for a more accurate interpretation of the capabilities of these models.

Summary

- The GraphGPS framework is at most as expressive (for graph distinguishability) as S-SEG-WL with S defined as in Theorem 12. This includes k-MIP-GT, Exphormer, and GPS++.
- When no positional encodings are used and there are no node and edge features, GraphGPS is at most as expressive as the 1-WL test.
- When Laplacian positional encodings are used, GraphGPS is strictly more expressive than the 1-WL test.
- Increasing the expressiveness of the positional encodings increases the upper bound on the expressiveness of GraphGPS.

5.2 Universal Approximation on Sequences

In this section, we show a universal approximation property of sequence-to-sequence Transformers based on the k-MIP self-attention mechanism, analogous to standard sequence-to-sequence Transformers. This result is crucial, as it guarantees that sparsifying the self-attention mechanism does not come at the cost of expressiveness, at least when an arbitrary number of layers can be used.

The buildup towards this result proceeds in three steps. First, we will define the class of functions that we aim to approximate and state the universal approximation property of standard sequence Transformers. While we will not use this approximation result, we provide it for context, as we will show that we can approximate exactly the same class of functions with k-MIP Sequence Transformers. Second, we will define the class of k-MIP Sequence Transformers, which are analogous to

standard Transformers but use the k-MIP instead of the full self-attention mechanism. Finally, we will state and prove the universal approximation property of k-MIP Sequence Transformers.

5.2.1 Set-up

We first define the class of functions that we aim to approximate with our k-MIP Sequence Transformers in Definition 13 below. This definition concerns functions that map sequences of tokens $(\boldsymbol{x}_i)_{i=1}^N$ into updated sequences $(\boldsymbol{x}'_i)_{i=1}^N$ of the same length, where the transformation is permutation equivariant w.r.t. the ordering of the tokens. While the permutation equivariance condition may seem unusual, this is exactly the type of functions that standard Transformers can represent when no positional encoding is applied to break the token symmetry. For convenience, instead of sequences, we use an equivalent matrix-based formulation, where the input and output are matrices $\boldsymbol{X}, \boldsymbol{X}' \in \mathbb{R}^{N\times d}$ (the 'tokens' being the rows of this matrix) and the operation is equivariant w.r.t. row permutations.

Definition 13. *Let $\mathcal{F}_{seq}$ be the class of functions $f : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ that satisfy the following conditions:*

- *f is continuous w.r.t. any entry-wise ℓ^p norm with $1 \leq p < \infty$.*
- *f has compact support in $\mathbb{R}^{N\times d}$.*
- *f is equivariant w.r.t. row permutations: for any permutation matrix $\boldsymbol{P} \in \{0,1\}^{N\times N}$, we have*

$$f(\boldsymbol{P}\boldsymbol{X}) = \boldsymbol{P}f(\boldsymbol{X}) \tag{5.5}$$

As discussed in section 2.4, [27] proved that standard full-attention Transformer networks can approximate any such function to arbitrary precision. Specifically, they first defined the class of full-attention Transformers $\mathcal{T}_{\text{full}}^{2,1,4}$ that consist of an arbitrary number of Transformer blocks, where each Transformer block contains 2 full-attention heads of width 1 and an MLP with width 4. Then, they showed the following theorem.

Theorem 14 (Theorem 2 from [27]). *Let $1 \leq p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{seq}$, there exists a full-attention Transformer $T \in \mathcal{T}_{full}^{2,1,4}$ such that*

$$\left(\int \|f(\boldsymbol{X}) - T(\boldsymbol{X})\|_p^p \, d\boldsymbol{X} \right)^{1/p} < \epsilon. \tag{5.6}$$

In what follows, we will show that exactly the same result holds for k-MIP Sequence Transformers, which use the k-MIP self-attention mechanism instead of the standard self-attention mechanism.

First, we define the class of k-MIP Transformer blocks, which compose to k-MIP Sequence Transformers. Note again that we write those sequence functions down with an equivalent matrix-based formulation, where the input and output are matrices $\boldsymbol{X}, \boldsymbol{X}' \in \mathbb{R}^{N \times d}$ (the 'tokens' being the rows of this matrix) and the operation is equivariant w.r.t. row permutations.

Definition 15 (From [27], modified to use the k-MIP attention mechanism). *A* k-MIP Transformer block $t_k^{h,m,r}$ *is a row-permutation equivariant mapping from $\mathbb{R}^{N \times d}$ to $\mathbb{R}^{N \times d}$ that implements the sequential operation*

$$t_k^{h,m,r} : \boldsymbol{X} \mapsto g(f(\boldsymbol{X})), \tag{5.7}$$

$$\textit{where} \quad f : \boldsymbol{X} \mapsto \boldsymbol{X} + \texttt{k-MIP-Attn}(\boldsymbol{X}), \tag{5.8}$$

$$\textit{and} \quad g : \boldsymbol{X} \mapsto \boldsymbol{X} + \texttt{MLP}(\boldsymbol{X}). \tag{5.9}$$

Here, h refers to the number of heads in `k-MIP-Attn`*, $m = d_{kq} = d_{val}$ is the hidden dimension of each head, and r is the hidden layer size of the MLP. Further,* `k-MIP-Attn` *and* `MLP` *are defined as in section 3.4 and section 4.1, respectively.*

This definition is equivalent to the one in [27], with the exception that `k-MIP-Attn` uses the top-k softmax function instead of the standard softmax function.

We now define k-MIP Sequence Transformers as a composition of k-MIP Transformer blocks.

Definition 16 (From [27], modified to use k-MIP Transformer blocks). *Let $\mathcal{T}_k^{h,m,r}$ be*

the class of k-MIP Sequence Transformers*, where each k-MIP Sequence Transformer is a composition of k-MIP Transformer blocks* $t_k^{h,m,r}$*.*

$$\mathcal{T}_k^{h,m,r} := \left\{ T_k : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d} \mid T_k \text{ is a composition of Transformer blocks } t_k^{h,m,r} \right\} \tag{5.10}$$

5.2.2 Results

The following theorem shows that k-MIP Transformers are universal approximators for continuous and permutation equivariant sequence-to-sequence functions. This result is completely analogous to Theorem 2 in [27], except from the use of the k-MIP self-attention mechanism instead of the full self-attention mechanism, which shows that the sparsification of the self-attention mechanism does not come at the cost of expressiveness, at least when an arbitrary number of layers can be used.

Theorem 17 (Theorem 2 from [27], adapted to use k-MIP Transformers)**.** *Let* $1 \leq p < \infty$ *and* $\epsilon > 0$*, then for any given* $f \in \mathcal{F}_{seq}$*, there exists a k-MIP Transformer* $T_k \in \mathcal{T}_k^{2,1,4}$ *such that*

$$\left(\int \|f(\boldsymbol{X}) - T_k(\boldsymbol{X})\|_p^p \, d\boldsymbol{X} \right)^{1/p} < \epsilon. \tag{5.11}$$

The proof of this theorem is based on the proof of Theorem 2 in [27] and can be found in subsection 5.4.2.

Note that every (standard) full-attention Transformer implements a continuous and permutation equivariant function w.r.t. the ordering of the tokens. Hence, if we restrict them to a compact domain, the class of functions that such Transformers can represent is contained in $\mathcal{F}_{\text{seq}}$. Consequently, Theorem 17 does not only show that k-MIP Sequence Transformers and standard Transformers can both universally approximate $\mathcal{F}_{\text{seq}}$, but also that the k-MIP Sequence Transformer can universally approximate *any* function that can be represented by a standard Transformer.

Further, if one removes the permutation equivariance constraint by using a positional encoding for sequences (like the sinusoidal encoding in [17]) to break the

token symmetry, an analogous result can be proven for universal approximation of any continuous function on sequences with compact support (without the permutation equivariance constraint), by both standard Transformers and k-MIP Sequence Transformers.

Finally, when using the node features of a graph as tokens, this k-MIP Sequence Transformer can already approximate any continuous function with compact support on this graph, if this graph is the empty graph. However, to approximate functions that incorporate the graph structure, positional encodings are necessary, as we will show in the next section.

Summary

- Sequence Transformers based on k-MIP self-attention are universal approximators for continuous and permutation equivariant functions.
- In this way, the k-MIP Sequence Transformer can approximate any function that can be represented by a standard Transformer.
- This result guarantees that the sparsification of the self-attention mechanism does not come at the cost of expressiveness.

5.3 Universal Approximation on Graphs

In this section, we extend the previous result for k-MIP Sequence Transformers to the graph setting. Specifically, we follow the approach of Exphormer [21] and SAN [28] and consider a graph Transformer to be a function from $\mathbb{R}^{N\times d} \times \mathbb{R}^{N\times N}$ to $\mathbb{R}^{N\times(d+N)}$ that takes in node features augmented with the weighted adjacency matrix and outputs updated node embeddings. This allows us to reuse the k-MIP sequence Transformer class $\mathcal{T}_k^{h,m,r}$ from section 5.2 as a class of graph Transformers.

Under the same assumptions as [21] and [28], we show that a k-MIP-GT can approximate any continuous function on graphs that is permutation equivariant w.r.t. the ordering of the nodes. This shows that sparsifying the self-attention mechanism does not come at the cost of expressiveness in the graph setting.

The buildup towards this result proceeds in two steps. First, we will define the class of functions that we aim to approximate and state the universal approximation property of the graph Transformers from Exphormer [21] and SAN [28]. While we will not use their approximation results, we provide them for context, because in the second step, we will show that we can approximate exactly the same class of functions with k-MIP-GTs.

5.3.1 Set-up

We first define the class of functions that we aim to approximate with the k-MIP-GT. These functions take in node features $\boldsymbol{X}$ and an adjacency matrix $\boldsymbol{A}$ and output updated node features.

Definition 18. *Let $\mathcal{F}_{graph}$ be the class of functions $f : \mathbb{R}^{N\times d} \times \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times(d+N)}$ that satisfy the following conditions:*

- *f is continuous w.r.t. any entry-wise ℓ^p norm with $1 \leq p < \infty$.*
- *f has compact support in $\mathbb{R}^{N\times d} \times \mathbb{R}^{N\times N}$.*
- *f is equivariant w.r.t. node index permutations: for any permutation matrix $\boldsymbol{P} \in \{0,1\}^{N\times N}$, we have*

$$f(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top) = \boldsymbol{P}f(\boldsymbol{X}, \boldsymbol{A}). \tag{5.12}$$

As discussed in section 2.4, [21] and [28] proved that their respective graph Transformers can approximate any such function to arbitrary precision. Specifically, SAN proved the theorem below for the case $d = 0$, but their result can be easily extended to $d > 0$.

Theorem 19 (Appendix C.2 from [28])**.** *Let $1 \leq p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{graph}$, there exist a matrix $\boldsymbol{M} \in \mathbb{R}^{N\times(d+N)}$ and a SAN node Transformer $T \in \mathcal{T}^{2,1,4}_{SAN}$ such that*

$$\left(\int\int \left\| f(\boldsymbol{X}, \boldsymbol{A}) - T_{SAN}\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{A}\end{bmatrix} + \boldsymbol{M}\right)\right\|_p^p d\boldsymbol{X}\, d\boldsymbol{A}\right)^{1/p} < \epsilon. \tag{5.13}$$

The theorem proven in the Exphormer paper [21] (Theorem E.3) is analogous, but they need to impose an extra assumption on their attention graph to achieve the same result.

In the next subsection, we will show that exactly the same result holds for k-MIP Graph Transformers.

Further, since each Transformer in $\mathcal{T}_k^{h,m,r}$ is a special case of the k-MIP Graph Transformer, where the MPNN layer is disabled and the adjacency matrix is used as a positional encoding for the node features, this result will imply that the k-MIP-GT can universally approximate the functions in $\mathcal{F}_{\text{graph}}$.

5.3.2 Results

The following theorem shows that k-MIP Transformers in $\mathcal{T}_k^{h,m,r}$ (see Definition 16) can approximate any continuous and permutation equivariant function on graphs.

Theorem 20. *Let $1 \leq p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{graph}$, there exist a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times (d+N)}$ and a k-MIP Transformer $T_k \in \mathcal{T}_k^{2,1,4}$ such that*

$$\left(\int \int \left\| f(\boldsymbol{X}, \boldsymbol{A}) - T_k \left(\begin{bmatrix} \boldsymbol{X} & \boldsymbol{A} \end{bmatrix} + \boldsymbol{M} \right) \right\|_p^p d\boldsymbol{X}\, d\boldsymbol{A} \right)^{1/p} < \epsilon. \tag{5.14}$$

The proof of this theorem can be found in subsection 5.4.3.

Since each Transformer in $\mathcal{T}_k^{h,m,r}$ is a special case of the k-MIP Graph Transformer, where the MPNN layer is disabled and the adjacency matrix is used as a positional encoding for the node features, this implies that the k-MIP-GT can universally approximate the functions in $\mathcal{F}_{\text{graph}}$. This result is completely analogous to the universal approximation results of Exphormer [21] and SAN [28], except that we use the k-MIP self-attention mechanism instead of the full self-attention mechanism. This shows that the sparsification of the self-attention mechanism does not come at the cost of expressiveness, at least when an arbitrary number of layers can be used.

5.3.3 Discussion

Theorem 20 provides the guarantee that our proposed k-MIP-GT can universally approximate any continuous function on graphs, under the assumption that the rows of the adjacency matrix are used as positional encodings and we allow for a matrix $\boldsymbol{M}$ to break the token symmetry. While these assumptions are not realistic in practice, as the rows of the adjacency matrix are usually not used as positional encoding (they break permutation equivariance) and the matrix $\boldsymbol{M}$ is not used in practice, full-attention graph Transformers need the same unrealistic assumptions to achieve the same result [21, 28]. Therefore, it is encouraging that a graph Transformer with our sparsified k-MIP self-attention mechanism still allows us to achieve the same expressivity results as full-attention graph Transformers.

Yet, we repeat our conclusion from section 5.1 that the expressive power of our method is fundamentally constrained by the positional and structural encodings applied to the node and edge features. If less expressive encodings were used than the adjacency matrix, some graphs would be no longer distinguishable by the model, so the functions that assign different values to these graphs could not be approximated. The same consideration applies to other graph Transformers and other instances of the GraphGPS framework.

The result in this section does not yet use the full power of the k-MIP-GT, as we have disabled the MPNN to prove Theorem 20. It would be interesting to investigate whether incorporating this component would allow to weaken the assumptions required to achieve universal approximation.

Further, the result in this section does not consider the approximation of functions that incorporate edge features. The approximation of such functions is not usually studied in the literature, and it was out of the scope of this project to investigate this. However, investigating to what extent graph Transformers and GNNs can approximate such functions would be an interesting direction for future research, to better understand the capabilities and limitations of models that use edge features.

Summary

- The k-MIP-GT is a universal approximator for continuous functions on graphs that are equivariant w.r.t. permutations of the ordering of the nodes, under the assumption that the adjacency matrix is used as a positional encoding and a matrix $\boldsymbol{M}$ is used to break the token symmetry.
- This result is analogous to the universal approximation results of full-attention graph Transformers [28] and operates under the same assumptions, showing that the sparsification of the self-attention mechanism does not come at the cost of expressiveness.

5.4 Proofs

In this section, we provide the proofs of the theorems previously stated in this chapter, including Theorems 12, 17 and 20. The proof for Theorem 12 borrows heavily from the definitions and concepts introduced in section 3.8.

This section is reasonably technical and may be skipped by readers who are not interested in the details of the proofs.

5.4.1 Proof of Theorem 12

To prove Theorem 12, we will first establish the following lemma. This lemma and its proof are based on Theorem 1 in [65], albeit substantial modifications were necessary to account for the fact that the GraphGPS framework iteratively computes edge embeddings in addition to node embeddings.

Lemma 21 (Theorem 1 in [65], modified)**.** *For any strongly regular structural encoding scheme $S = (f_A, f_R)$ and graph $G = (V, E)$ with node features $\boldsymbol{X}$, if a graph neural model $\mathcal{A}$ that computes node embeddings $d^{(l)} : V \to \mathcal{C}$ and node pair embeddings $e^{(l)} : V \times V \to \mathcal{C}$ satisfies the following conditions:*

C1. $\mathcal{A}$ computes the initial node and node pair embeddings with

$$d^{(0)}(v) = \phi(\boldsymbol{X}_v, f_A(v, G)) \tag{5.15}$$

$$e^{(0)}(r, v) = \psi(f_R(v, r, G)) \tag{5.16}$$

where ϕ and ψ are model-specific functions.

C2. $\mathcal{A}$ aggregates and updates node and edge embeddings iteratively with

$$d^{(l)}(v) = \sigma\left(\left\{\!\!\left\{ \left(d^{(l-1)}(r), e^{(l-1)}(r, v), f_R(v, r, G)\right) \mid r \in V \right\}\!\!\right\}\right) \tag{5.17}$$

$$e^{(l)}(r, v) = \tau\left(e^{(l-1)}(r, v), d^{(l-1)}(r), d^{(l-1)}(v), f_R(v, r, G)\right) \tag{5.18}$$

where σ and τ are model-specific functions.

Then any two graphs G_1 and G_2 that are distinguished by $\mathcal{A}$ in iteration l are also distinguished by S-SEG-WL in iteration l.

Proof. We consider two not necessarily distinct graphs $G_v = (V_v, E_v, \boldsymbol{X}_v, \boldsymbol{E}_v, \boldsymbol{A}_v)$ and $G_w = (V_w, E_w, \boldsymbol{X}_w, \boldsymbol{E}_w, \boldsymbol{A}_w)$ and denote the colour given by S-SEG-WL to node $u \in V_v \cup V_w$ at iteration l by $c^{(l)}(u)$. We first show by induction that for any iteration l and any nodes $v, r \in V_v, w, s \in V_w$, the following two implications hold:

$$c^{(l)}(v) = c^{(l)}(w) \implies d^{(l)}(v) = d^{(l)}(w) \tag{IH.1}$$

$$\left.\begin{aligned} c^{(l)}(r) &= c^{(l)}(s) \\ f_R(v, r, G_v) &= f_R(w, s, G_w) \\ c^{(l)}(v) &= c^{(l)}(w) \end{aligned}\right\} \implies e^{(l)}(r, v) = e^{(l)}(s, w) \tag{IH.2}$$

Base case For $l = 0$, (IH.1) holds because $c^{(0)}(v) = c^{(0)}(w)$ implies $\boldsymbol{X}_v = \boldsymbol{X}_w$ and $f_A(v, G_v) = f_A(w, G_w)$, which leads to $d^{(0)}(v) = d^{(0)}(w)$. Further, (IH.2) holds because $f_R(v, r, G_v) = f_R(w, s, G_w)$ directly entails $e^{(0)}(r, v) = e^{(0)}(s, w)$.

Induction step for (IH.1) Suppose that the induction hypotheses (IH.1) and (IH.2) hold for iteration l and that $c^{(l+1)}(v) = c^{(l+1)}(w)$. From the injectivity of

function Φ, we have

$$\{\!\{ \big(c^{(l)}(r), f_R(v, r, G_v)\big) \mid r \in V_v \}\!\} = \{\!\{ \big(c^{(l)}(s), f_R(w, s, G_w)\big) \mid s \in V_w \}\!\} \tag{5.19}$$

Because f_R is strongly regular, we can choose a discriminator function ξ such that for all graphs G and nodes a, b, $\xi(f_R(a, b, G), G) = \mathbb{1}_{a=b}$. By applying ξ to the second element of each tuple in both multisets together with the corresponding graph, we obtain

$$\{\!\{ \big(c^{(l)}(r), \mathbb{1}_{r=v}\big) \mid r \in V_v \}\!\} = \{\!\{ \big(c^{(l)}(s), \mathbb{1}_{s=w}\big) \mid s \in V_w \}\!\} \tag{5.20}$$

In each of the above multisets, there is a unique tuple for which the second element is 1, namely the tuples corresponding to $r = v$ and $s = w$, respectively. Therefore, these tuples must be equal, which implies

$$c^{(l)}(v) = c^{(l)}(w) \tag{5.21}$$

Because the two multisets in (5.19) are identical and finite, their elements can be matched in pairs. Further, by the induction hypotheses and the fact that $c^{(l)}(v) = c^{(l)}(w)$, we have that for any $r \in V_v$ and $s \in V_w$, $\big(c^{(l)}(r), f_R(v, r, G_v)\big) = \big(c^{(l)}(s), f_R(w, s, G_w)\big)$ implies $\big(d^{(l)}(r), e^{(l)}(r, v), f_R(v, r, G_v)\big) = \big(d^{(l)}(s), e^{(l)}(s, w), f_R(w, s, G_w)\big)$. Hence,

$$\begin{aligned} &\{\!\{ \big(d^{(l)}(r), e^{(l)}(r, v), f_R(v, r, G_v)\big) \mid r \in V_v \}\!\} \\ &= \{\!\{ \big(d^{(l)}(s), e^{(l)}(s, w), f_R(w, s, G_w)\big) \mid s \in V_w \}\!\} \end{aligned} \tag{5.22}$$

Considering $\mathcal{A}$ updates node labels according to (5.17), $d^{(l+1)}(v) = d^{(l+1)}(w)$ holds. This completes the induction step for (IH.1).

Induction step for (IH.2) While retaining our assumptions that the induction hypotheses (IH.1) and (IH.2) hold for iteration l and that $c^{(l+1)}(v) = c^{(l+1)}(w)$, suppose additionally that $c^{(l+1)}(r) = c^{(l+1)}(s)$ and $f_R(v, r, G_v) = f_R(w, s, G_w)$. Analogously to the derivation preceding (5.21), it follows from $c^{(l+1)}(v) = c^{(l+1)}(w)$ that $c^{(l)}(v) = c^{(l)}(w)$ and from $c^{(l+1)}(r) = c^{(l+1)}(s)$ that $c^{(l)}(r) = c^{(l)}(s)$. Induction hy-

pothesis (IH.1) now yields $d^{(l)}(v) = d^{(l)}(w)$ and $d^{(l)}(r) = d^{(l)}(s)$, while induction hypothesis (IH.2) leads to $e^{(l)}(r, v) = e^{(l)}(s, w)$. Combining these results, we have

$$\begin{aligned} &\big(e^{(l)}(r, v), d^{(l)}(r), d^{(l)}(v), f_R(v, r, G_v)\big) \\ &= \big(e^{(l)}(s, w), d^{(l)}(s), d^{(l)}(w), f_R(w, s, G_w)\big) \end{aligned} \tag{5.23}$$

And because $\mathcal{A}$ updates edge labels according to (5.18), we have $e^{(l+1)}(r, v) = e^{(l+1)}(s, w)$. This completes the induction step for (IH.2), and thereby the proof that (IH.1) and (IH.2) hold for all iterations l.

Consequence of (IH.1) Now consider two graphs G_1 and G_2 that are distinguished by $\mathcal{A}$ after l iterations. We will prove by contradiction that G_1 and G_2 are also distinguished by the S-SEG-WL test after l iterations. Suppose that G_1 and G_2 are not distinguished by the S-SEG-WL test after l iterations, i.e.

$$\big\{\!\big\{c^{(l)}(v) \mid v \in V_1\big\}\!\big\} = \big\{\!\big\{c^{(l)}(w) \mid w \in V_2\big\}\!\big\} \tag{5.24}$$

Because the two multisets in (5.24) are identical and finite, their elements can be matched in pairs of equal elements. By (IH.1), we have that for any $v \in V_1, w \in V_2$, $c^{(l)}(v) = c^{(l)}(w)$ implies $d^{(l)}(v) = d^{(l)}(w)$, from which

$$\big\{\!\big\{d^{(l)}(v) \mid v \in V_1\big\}\!\big\} = \big\{\!\big\{d^{(l)}(w) \mid w \in V_2\big\}\!\big\} \tag{5.25}$$

This would imply that G_1 and G_2 are not distinguished by $\mathcal{A}$ after l iterations, which contradicts our assumption. Therefore, G_1 and G_2 must be distinguished by the S-SEG-WL test after l iterations. □

Now that Lemma 21 is proven, we can proceed with the proof of Theorem 12, which we restate here for convenience.

Theorem 12 (Restated)**.** *Let $\mathcal{A}$ be an instance of the GraphGPS framework that enhances its node features with $\nu(\boldsymbol{A})$ and its edge features with $\mu(\boldsymbol{A})$. Then $\mathcal{A}$ can only distinguish graphs that are also distinguishable by the S-SEG-WL test, where*

$S = (f_A, f_R)$ and f_A, f_R are defined as

$$f_A(v, G) = \nu(\boldsymbol{A})_v, \tag{5.1}$$

$$f_R(v, u, G) = \begin{cases} (0, \mathbb{1}_{u=v}, \mathbf{0}_{d_{edge}}, \mathbf{0}_{d_{PE}}) & \text{if } (u, v) \notin E \\ (1, \mathbb{1}_{u=v}, \boldsymbol{E}_{uv}, \mu(\boldsymbol{A})_{uv}) & \text{if } (u, v) \in E \end{cases}, \tag{5.2}$$

where $\mathcal{C} = \mathbb{R}^{d_{PE}} \cup \{0, 1\}^2 \times \mathbb{R}^{d_{edge}} \times \mathbb{R}^{d_{PE}} \cup \mathbb{R}^d \cup \mathbb{R}^{d_{edge}}$.

Proof. Let $\mathcal{B}$ be an instance of the GraphGPS framework (section 3.7) that iteratively generates the node and edge embeddings $(\boldsymbol{X}^{(l)})_{l=0}^L$ and $(\boldsymbol{E}^{(l)})_{l=0}^L$. Extend $\mathcal{B}$ to the colouring algorithm $\mathcal{A}$ that iteratively computes node and node pair colours as follows:

$$d^{(l)}(v) = \boldsymbol{X}_v^{(l)} \tag{5.26}$$

$$e^{(l)}(r, v) = \begin{cases} \mathbf{0}_d & \text{if } (r, v) \notin E \\ \boldsymbol{E}_{rv}^{(l)} & \text{if } (r, v) \in E \end{cases} \tag{5.27}$$

First, note that the structural encoding scheme in the theorem statement is strongly regular, as any function ξ for which $\xi(t, G) \coloneqq t.\texttt{second}$ if t is a 4-tuple is a discriminator function for f_R. We will now prove that $\mathcal{A}$ satisfies the conditions of Lemma 21, where $S = (f_A, f_R)$ and $\mathcal{C}$ are as defined in the theorem statement. Once this is established, it follows that $\mathcal{A}$ can only distinguish graphs that are also distinguishable by the S-SEG-WL test. The desired result then follows from the observation that $\mathcal{A}$ and $\mathcal{B}$ always generate the same node colourings, and thus distinguish exactly the same graphs.

$\mathcal{A}$ satisfies Condition C1 by construction: the initial node and node pair colours are

computed as

$$d^{(0)}(v) = \boldsymbol{X}_v^{(0)} = \begin{bmatrix} \boldsymbol{X}_v \\ \rho(\boldsymbol{A})_v \end{bmatrix} = \phi(\boldsymbol{X}_v, f_A(v, G)) \tag{5.28}$$

$$e^{(0)}(r, v) = \begin{cases} \mathbf{0}_d & \text{if } (r, v) \notin E \\ \begin{bmatrix} \boldsymbol{E}_{rv} \\ \mu(\boldsymbol{A})_{rv} \end{bmatrix} & \text{if } (r, v) \in E \end{cases} = \psi(f_R(v, r, G)) \tag{5.29}$$

if ϕ is chosen to be the concatenation operation and ψ is chosen to be a function that concatenates the last two elements when given a 4-tuple.

$\mathcal{A}$ also satisfies Condition C2. To see this, we will describe the construction of the functions σ and τ such that the update rule of $\mathcal{A}$ can be written as (5.17) and (5.18).

σ takes in $I_v = \left\{\!\!\left\{ \left(\boldsymbol{X}_r^{(l-1)}, e^{(l-1)}(r, v), f_R(v, r, G) \right) \mid r \in V \right\}\!\!\right\}$ and outputs $d^{(l)}(v) = \boldsymbol{X}_v^{(l)}$. Given the input I_v, first retrieve the node feature vector $\boldsymbol{X}_v^{(l-1)}$ by selecting the unique tuple $t \in I_v$ for which $\mathbb{1}_{r=v} = f_R(v, r, G)\texttt{.second} = 1$ and letting $\boldsymbol{X}_v^{(l-1)} = t\texttt{.first}$. Then, retrieve the multiset $\left\{\!\!\left\{ (\boldsymbol{X}_r^{(l-1)}, \boldsymbol{E}_{rv}^{(l-1)}) \mid (r, v) \in E \right\}\!\!\right\}$ from I_v by selecting $\boldsymbol{X}_r^{(l-1)}$ and $\boldsymbol{E}_{rv}^{(l-1)}$ from all tuples t for which $\mathbb{1}_{(r,v)\in E} = f_R(v, r, G)\texttt{.first} = 1$. Because of Assumption A1, we now have all the necessary inputs to apply the message passing layer `MPNN` and obtain $\boldsymbol{X}_{M,v}^{*(l)}$. Further, compute $\boldsymbol{X}_{M,v}^{(l)} = \boldsymbol{X}_{M,v}^{*(l)} + \boldsymbol{X}_v^{(l-1)}$. Next, retrieve the multiset $\left\{\!\!\left\{ \boldsymbol{X}_r^{(l-1)} \mid r \in V \right\}\!\!\right\}$ from I_v by selecting the first element of all tuples. Assumption A3 guarantees that this and $\boldsymbol{X}_v^{(l-1)}$ is all we need to determine $\texttt{Attn}(\boldsymbol{X}^{(l-1)})_v$. Thus, we can compute $\boldsymbol{X}_{T,v}^{(l)} = \texttt{Attn}(\boldsymbol{X}^{(l-1)})_v + \boldsymbol{X}_v^{(l-1)}$. Finally, Assumption A4 ensures that `MLP` computes an elementwise function, so we can compute $\boldsymbol{X}_v^{(l)} = \texttt{MLP}(\boldsymbol{X}_{M,v}^{(l)} + \boldsymbol{X}_{T,v}^{(l)}) + \boldsymbol{X}_{M,v}^{(l)} + \boldsymbol{X}_{T,v}^{(l)}$. By following this procedure, σ can implement the desired update rule.

τ takes in $I_{rv} = \left(e^{(l-1)}(r, v), d^{(l-1)}(r), d^{(l-1)}(v), f_R(v, r, G)\right)$ and outputs $e^{(l)}(r, v)$. Given the input I_{rv}, first determine whether $(r, v) \in E$ by checking the third element of $f_R(v, r, G)$. If $(r, v) \notin E$, output $\mathbf{0}_d$. Otherwise, Assumption A2 guarantees that $e^{(l)}(r, v) = \boldsymbol{E}_{rv}^{(l)}$ depends only on $e^{(l-1)}(r, v)$, $d^{(l-1)}(r)$ and $d^{(l-1)}(v)$. Thus, τ can be implemented as desired.

This completes the proof that $\mathcal{A}$ satisfies the conditions of Lemma 21, where $S = (f_A, f_R)$ and $\mathcal{C}$ are as defined in the theorem statement. It follows that $\mathcal{A}$ can only distinguish graphs that are also distinguishable by the (f_A, f_R)-SEG-WL test. Now observe that $\mathcal{A}$ and $\mathcal{B}$ always generate the same node colourings, and thus distinguish exactly the same graphs. From this follows the desired result. □

5.4.2 Proof of Theorem 17

In this subsection, we provide the proof of Theorem 17, which we restate here for convenience.

Theorem 17 (Restated). *Let $1 \leq p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{seq}$, there exists a k-MIP Transformer $T_k \in \mathcal{T}_k^{2,1,4}$ such that*

$$\left(\int \|f(\boldsymbol{X}) - T_k(\boldsymbol{X})\|_p^p \, d\boldsymbol{X} \right)^{1/p} < \epsilon. \tag{5.11}$$

Proof. The proof of this theorem is completely analogous to the proof of Theorem 2 in [27]. The only difference that needs to be accounted for is the use of the top-k softmax function `top-k-softmax` instead of the standard softmax function `softmax`. However, the only property of the softmax function that is used in the proof is that it can be made arbitrarily close to a hardmax function `hardmax` by scaling up the input matrix $\boldsymbol{Z}$ [27]:

$$\texttt{softmax}[\lambda \boldsymbol{Z}] \to \texttt{hardmax}[\boldsymbol{Z}] \text{ as } \lambda \to \infty. \tag{5.30}$$

This property also holds for the top-k softmax function `top-k-softmax`, which makes the proof valid for our modified definition of the Transformer class $\mathcal{T}_k^{2,1,4}$. □

5.4.3 Proof of Theorem 20

To prove Theorem 20, we will first establish the following lemma.

Lemma 22 (Based on Theorem 3 from [27]). *Let $1 \leq p < \infty$ and $\epsilon > 0$. Further, let f be an arbitrary function from $\mathbb{R}^{N \times d}$ to $\mathbb{R}^{N \times d}$ that (1) is continuous w.r.t.*

the entry-wise ℓ^p norm, and (2) has compact support. Then, there exist a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times d}$ and a k-MIP Transformer $T_k \in \mathcal{T}_k^{2,1,4}$ such that

$$\left(\int_{\boldsymbol{X} \in \mathbb{R}^{N \times d}} \| f(\boldsymbol{X}) - T_k(\boldsymbol{X} + \boldsymbol{M}) \|_p^p \, d\boldsymbol{X} \right)^{1/p} < \epsilon \tag{5.31}$$

Proof. The proof is completely analogous to the proof of Theorem 3 in [27]. The only difference that needs to be accounted for is the use of the top-k softmax function `top-k-softmax` instead of the standard softmax function `softmax`. However, the only property of the softmax function that is used in the proof is that it can be made arbitrarily close to a hardmax function `hardmax` by scaling up the input matrix $\boldsymbol{Z}$ [27]:

$$\texttt{softmax}[\lambda \boldsymbol{Z}] \to \texttt{hardmax}[\boldsymbol{Z}] \text{ as } \lambda \to \infty \tag{5.32}$$

This property also holds for the top-k softmax function `top-k-softmax`, which makes the proof valid for our modified definition of the Transformer class $\mathcal{T}_k^{2,1,4}$. □

Now, we can continue with the proof of Theorem 20, which we restate here for convenience.

Theorem 20 (Restated)**.** *Let $1 \leq p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{graph}$, there exist a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times (d+N)}$ and a k-MIP Transformer $T_k \in \mathcal{T}_k^{2,1,4}$ such that*

$$\left(\int \int \left\| f(\boldsymbol{X}, \boldsymbol{A}) - T_k\left(\begin{bmatrix} \boldsymbol{X} & \boldsymbol{A} \end{bmatrix} + \boldsymbol{M} \right) \right\|_p^p d\boldsymbol{X} \, d\boldsymbol{A} \right)^{1/p} < \epsilon. \tag{5.14}$$

Proof. Fix any function $f : \mathbb{R}^{N \times d} \times \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times (d+N)}$ that satisfies the assumptions of the theorem. Then we can construct a function $g : \mathbb{R}^{N \times (d+N)} \to \mathbb{R}^{N \times (d+N)}$ as follows:

$$g\left(\begin{bmatrix} \boldsymbol{X} & \boldsymbol{A} \end{bmatrix} \right) = f(\boldsymbol{X}, \boldsymbol{A}) \tag{5.33}$$

Like f, this function g is (1) continuous w.r.t. the entry-wise ℓ^p norm, (2) has compact support, and (3) is equivariant w.r.t. node index permutations. Therefore, it satisfies the conditions of Lemma 22. Consequently, there exist a matrix $\boldsymbol{M} \in$

$\mathbb{R}^{N\times(d+N)}$ and a k-MIP Transformer $T_k \in \mathcal{T}_k^{2,1,4}$ such that

$$\begin{aligned}
&\left(\int_{\boldsymbol{X}}\int_{\boldsymbol{A}} \left\| f(\boldsymbol{X},\boldsymbol{A}) - T_k\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{A}\end{bmatrix} + \boldsymbol{M}\right)\right\|_p^p d\boldsymbol{X}\, d\boldsymbol{A}\right)^{1/p} \\
&\left(\int_{\boldsymbol{X}}\int_{\boldsymbol{A}} \left\| g\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{A}\end{bmatrix}\right) - T_k\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{A}\end{bmatrix} + \boldsymbol{M}\right)\right\|_p^p d\boldsymbol{X}\, d\boldsymbol{A}\right)^{1/p} \\
&= \left(\int_{\boldsymbol{Z}\in\mathbb{R}^{N\times(d+N)}} \left\| g\left(\boldsymbol{Z}\right) - T_k\left(\boldsymbol{Z}+\boldsymbol{M}\right)\right\|_p^p d\boldsymbol{Z}\right)^{1/p} < \epsilon
\end{aligned} \tag{5.34}$$

□

6 | Experimental Results

The primary objective of this chapter is to demonstrate the scalability and efficiency of our k-MIP Graph Transformer architecture on datasets with increasingly larger graphs. We build towards this goal through a systematic progression of experiments and analyses.

We start by establishing the foundational performance of our model on small-graph datasets in section 6.1. Then, in sections 6.2 through 6.5, we investigate the efficiency and characteristics of the k-MIP Graph Transformer in these small-scale regimes. These initial experiments help us understand the capabilities and limitations of our approach, providing critical insights for scaling up. In section 6.6, we apply these lessons to train and evaluate the k-MIP Graph Transformer on datasets with graphs of up to 500K nodes. Finally, sections 6.7 and 6.8 conclude with an in-depth analysis of the trained models, offering further insights into the results.

In detail, the structure of this chapter is as follows.

Performance on various small-graph datasets (section 6.1) As a proof of concept, we evaluate the k-MIP Graph Transformer on four datasets with small graphs. We compare its performance against other efficient graph Transformers and more traditional graph neural networks to establish its effectiveness.

Efficiency (section 6.2) We compare the efficiency of full self-attention, k-MIP attention with naive brute-force search, and k-MIP attention with symbolic matrices. In particular, we demonstrate how we achieve a speedup of two orders of magnitude over full attention.

Negative Result: Ball Tree Search (section 6.3) We discuss our attempt to improve the computational complexity of the k-MIP search, and explain why it was not useful for practical purposes.

Influence of k (section 6.4) We investigate how the number of keys attended to per query influences the performance and efficiency of the k-MIP Graph Transformer.

Influence of d_{kq} (section 6.5) We investigate how the dimensionality of the key-query space d_{kq} influences the performance of the k-MIP Graph Transformer.

Scaling to larger datasets (section 6.6) We train the k-MIP Graph Transformer on four datasets with increasingly larger graphs (up to 500K nodes), and evaluate its performance and efficiency.

An approximation for full attention? (section 6.7) We investigate to what extent k-MIP self-attention can be regarded as an approximation to full self-attention.

Inspecting the Attention Graphs (section 6.8) We analyse the characteristics of the attention graphs generated by the k-MIP Graph Transformer.

6.1 Performance on Various Small-Graph Datasets

In this section, we evaluate the empirical performance of our k-MIP Graph Transformer on a variety of small-graph datasets including both graph classification and node classification tasks. We demonstrate that, on these datasets, its performance is competitive with state-of-the-art efficient graph Transformers. This is a necessary first step to establish the effectiveness of our model before scaling it up to larger datasets.

Datasets The chosen datasets are Cifar10 [86], MalNet-Tiny [66, 67], PascalVOC-SP [86], and Peptides-Func [86], which are described in more detail in section A.1. These were chosen to represent a diversity of graph sizes, graph types, and tasks.

Baselines We compare the performance of our model against four graph Transformer models and four traditional graph neural network architectures. The traditional graph neural networks are Graph Convolutional Network (GCN) [33], Graph Isomorphism Network with Edge Features (GINE) [84], Graph Attention Network

(GAT) [34], and Gated Graph Convolutional Network (GatedGCN) [85]. The graph Transformer baselines include Exphormer [21] – which is based on the GraphGPS framework – as well as the GraphGPS framework with three other attention mechanisms as their Global Attention Layer. The three other attention mechanisms are BigBird [52], Performer [46], and the vanilla Transformer [17]. BigBird is a The choice for these graph Transformer baselines is based on two key reasons. First, they serve as strong baselines, as both the GraphGPS framework and the Exphormer architecture achieved state-of-the-art results at the time of their publications in 2022 and 2023, respectively, including on several of the datasets we use in this section. Second, by comparing k-MIP-GT to other instances of the GraphGPS framework using different attention mechanisms, we can effectively assess the performance of the k-MIP self-attention mechanism in comparison to other attention approaches.

Set-up For each dataset/model combination, we train five models with different random seeds. For each such run, we save the test performance at the epoch where the validation performance is the best, where the performance in both cases is given by the metric of interest for that dataset. In Table 6.1, we report the mean and standard deviation of the test performances across the five runs.

Due to resource constraints, it was not possible to conduct an extensive hyperparameter search for each model across all datasets. Instead, we selected the hyperparameters based on the hyperparameters reported in the Exphormer paper [21]. Specifically, for the graph Transformer models, including Exphormer and k-MIP-GT, we used the same hyperparameters as those used for Exphormer in their paper, with minor modifications to satisfy an equal parameter budget for each model. For the GNN baselines, we used the configurations from the Exphormer repository when available, and otherwise extrapolated the configurations from other models and datasets. A detailed list of the hyperparameters and the rationale behind their selection can be found in section A.2.

Results and Discussion Table 6.1 shows the mean and standard deviation of the test performance across five runs for each method/dataset combination. We

Table 6.1: Test performance of k-MIP-GT and baselines on four small-graph datasets. Shown is the mean ± std over five runs. The **first** and **second** best results are highlighted.

Model/Dataset	Cifar10 Accuracy ↑	MalNet-Tiny Accuracy ↑	PascalVOC-SP F1 score ↑	Peptides-Func AP ↑
GCN	52.23 ± 0.40	92.88 ± 0.39	12.90 ± 0.76	64.87 ± 0.36
GINE	57.00 ± 1.18	89.26 ± 1.00	13.44 ± 0.40	58.41 ± 0.52
GAT	55.29 ± 0.35	91.22 ± 1.32	14.88 ± 0.56	62.49 ± 0.32
GatedGCN	69.24 ± 0.17	93.08 ± 0.50	30.09 ± 1.60	63.87 ± 0.15
GPS + BigBird	74.49 ± 0.48	92.82 ± 0.54	32.89 ± 0.82	62.95 ± 0.24
GPS + Performer	69.84 ± 0.47	92.38 ± 0.55	35.96 ± 1.13	58.69 ± 0.85
GPS + Transformer	**76.27 ± 0.52**	92.60 ± 1.00	38.22 ± 1.36	**66.00 ± 0.48**
Exphormer	74.46 ± 0.46	**93.82 ± 0.65**	**39.90 ± 0.96**	64.25 ± 0.54
k-MIP-GT	**75.20 ± 0.34**	**93.34 ± 0.89**	**38.53 ± 1.26**	**65.66 ± 0.41**

make three observations based on these results.

First, we observe that our k-MIP Graph Transformer is competitive with the baselines, getting a second place in all four datasets. This is a strong result, as those graph Transformer baselines were state-of-the-art as little as a year ago.

Second, the fact that the k-MIP-GT – itself an instance of the GraphGPS framework – outperforms GPS+BigBird and GPS+Performer, while being roughly on par with GPS+Transformer, demonstrates the effectiveness of the k-MIP self-attention mechanism.

Third, we observe that the hyperparameters tuned for Exphormer improved the results of almost all graph Transformer models compared to the results reported in the GraphGPS paper [67]. The only exceptions were the GPS+Performer model on all four datasets and the GPS+Transformer model on MalNet-Tiny, but in both cases they used a considerably larger parameter budget than we did.

Moreover, the difference for the baseline models is even more significant. For instance, where the GraphGPS paper reports only a 59.30% average precision for GCN on the Peptides-Func dataset, we achieved an average precision of 64.87%. This is in line with the results obtained by a recent paper [104] that criticized the hyperparameter choices in the GraphGPS paper, and showed that the performance gap between graph Transformers and traditional GNNs is smaller than previously thought. When we convert this dissertation into a paper, we plan to do a more

extensive (and resource-intensive) hyperparameter search, to ensure a near-optimal performance for all models on all datasets and potentially improve the performance of our own method.

To conclude, in this section we have evaluated the performance of the k-MIP Graph Transformer on four small-graph datasets. We have demonstrated that its performance is competitive with state-of-the-art graph Transformer models, paving the way for a successful scale-up to larger datasets. In the following sections, we will investigate various properties of the k-MIP Graph Transformer, which will provide insight into how to make the method scalable to datasets with larger graphs.

6.2 Efficiency

When attempting to scale graph Transformers to large graphs, the attention mechanism quickly becomes a computational bottleneck due to its quadratic complexity. In this section, we evaluate to what extent k-MIP self-attention is more efficient than full self-attention, and how the efficiency of the k-MIP search can be improved using symbolic matrices. Specifically, we investigate the following research question:

Research question How do the efficiencies of full self-attention, k-MIP attention with naive brute-force k-MIPS, and k-MIP attention with k-MIPS using symbolic matrices compare?

Set-up To study this research question, we set up an experiment where we measure the runtime of each approach for various values of the number of nodes, in both an inference and a training setting. Specifically, in each run we sample normally distributed key, query and value matrices of size $N \times d_{kq}$, for each $N \in \{100 \cdot \sqrt{10}^i \mid i = 0, \dots, 6\}$ and $d_{kq} = 10$. Here, N is the number of tokens and d_{kq} is the dimensionality of the key-query space. We then run an attention pass with these keys, queries, and values, using all three approaches in both an inference setting (forward pass only, no gradient tracking required) and a training setting (forward and backward pass). We repeat this experiment 6 times, exclude the first "cold"

run, record the time taken by each approach, and plot the means in Figure 6.1 and Figure 6.2 for the training setting. This experiment is run on an Intel® Core™ i7-9750H Processor and a NVIDIA GeForce RTX 2060. The time is measured in seconds.

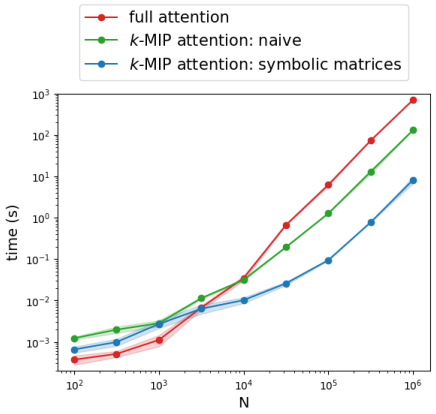
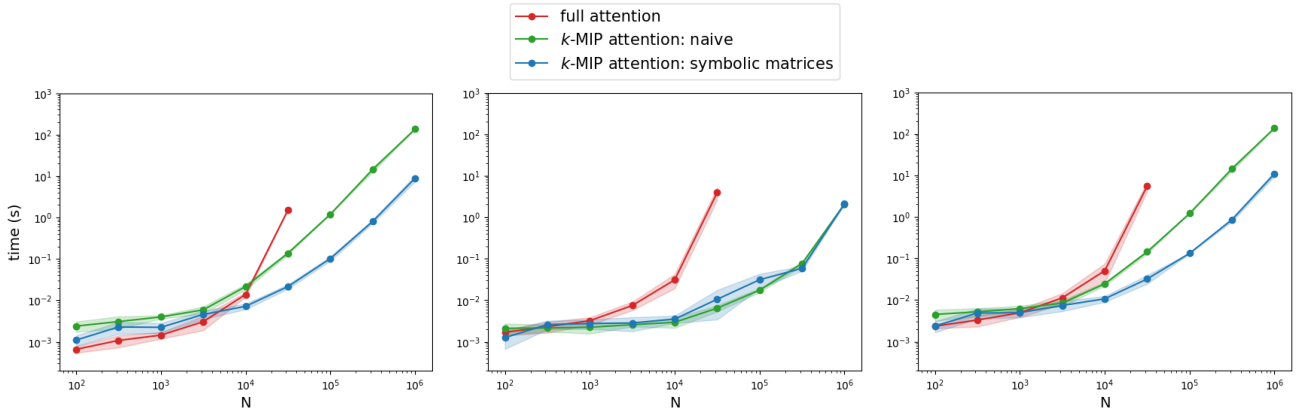


Figure 6.1: Efficiency comparison in an inference setting of (1) full self-attention, (2) k-MIP attention with naive brute-force k-MIPS, and (3) k-MIP attention with brute-force k-MIPS with symbolic matrices. Shown is the mean ± std over five runs.



(a) Forward Pass (b) Backward Pass (c) Total Iteration Time

Figure 6.2: Efficiency comparison in a training setting of (1) full attention, (2) k-MIP attention with naive brute-force k-MIPS, and (3) k-MIP attention with brute-force k-MIPS with symbolic matrices. Shown is the mean ± std over five runs. Full attention gave out-of-memory errors for $N \geq 10^5$.

Naive k-MIP Attention vs. Full Attention We begin by comparing the efficiency and scalability of full attention and naive k-MIP attention.

- **Inference Setting:** For smaller values of N, full attention outperforms naive k-MIP attention due to the overhead associated with k-MIP search, such as

identifying the top k indices and gathering the keys and values corresponding to these indices. However, this difference is not critical, as attention typically is not the bottleneck for low N. As N grows, the balance shifts, and by the time N reaches 10^4, naive k-MIP attention becomes the more efficient method. This is because k-MIP attention needs to perform softmax functions over only k elements per query instead of N, and aggregate k weighted value vectors per query instead of N. This leads to a *speedup of a factor 5* for the inference setting, as shown in Figure 6.1.

- **Training - Forward Pass:** In the training setting, full attention must store the intermediate activations for each of the N^2 inner products (for the gradient computation in the backward pass), which causes out-of-memory errors for $N \geq 10^5$. Scaling further becomes impractical: for reference, storing activations for a single attention head when $N = 10^6$ would require 4TB of GPU memory. k-MIP attention, on the other hand, only needs to store activations for Nk inner products, significantly reducing the memory footprint. While the attention score matrix is still materialised in global GPU memory for performing k-MIPS, it does not need to be stored at once. This makes k-MIP attention *applicable to much larger graphs*, and (because memory access is slow) also yields a *speedup of a factor 10* for the largest N.

- **Training - Backward Pass:** During the backward pass, the efficiency gap is even more pronounced. Where full attention must compute intermediate gradients for all N^2 inner products during the backward pass, k-MIP attention computes gradients for only Nk inner products[1]. In theory, this leads to a linear computational complexity of the backward pass. While this initially linear regime breaks down at $N = 10^6$ (see Figure 6.2b), the backward pass is still *dramatically more efficient* than for full attention. This makes the forward pass the bottleneck for the training setting.

- **Training - Total Time:** Where the backward pass dominates the total iter-

[1] In particular, no gradients need to be computed for the N^2 inner products used to find the top k keys, as these are not used in the backward pass.

ation time for full attention (roughly 65% of total runtime for $N = 10^{4.5}$), the forward pass dominates the total iteration time for k-MIP attention (roughly 98% of total runtime for $N = 10^6$) thanks to the linear complexity of the backward pass. In total, using k-MIP attention instead of full attention leads to a *factor 50 speedup* for large N, as well as a *linear memory footprint*, which allows for training on much larger graphs.

- **Summary:** During inference, both mechanisms need to compute the full attention matrix. However, the fact that k-MIP attention only needs to perform softmax functions over k elements, and aggregate k weighted value vectors per query, leads to a speedup of a factor 5 for large N. During training, full attention needs to store intermediate activations for all N^2 inner products during the forward pass and compute the gradients for all N^2 inner products during the backward pass. This leads to out-of-memory errors for $N \geq 10^{4.5}$ and a slow backward pass. Because k-MIP attention only needs to store the activations and compute the gradients of the Nk inner products corresponding to the top k keys for each query, it has a linear memory footprint and yields a speedup of a factor 50 for large N.

k-MIP Attention with Symbolic Matrices vs. Naive k-MIP Attention Now, we investigate how the use of symbolic matrices can further improve the efficiency of the k-MIP search.

- **Forward Pass:** The use of symbolic matrices with PyKeops [95] makes sure that the k-MIP search is performed without ever sending intermediate results to the GPU's global memory. This leads to a *speedup of a factor 10-15* for the forward pass, in both the inference and training settings. Further, the only GPU global memory that is used is for storing the inputs and results of the k-MIP search, which allows for a *negligible memory footprint*.
- **Backward Pass:** Since no gradients need to be propagated to the inner products that were computed to find the top k keys, the use of symbolic matrices has no effect on the efficiency of the backward pass, which the results in Figure 6.2b

confirm. Hence, the backward pass is very efficient for both methods.

- **Training - Total Time:** Since the k-MIP search in the forward pass is the bottleneck when $N \geq 10^3$, using symbolic matrices leads to a *speedup of a factor 10* over naive k-MIP attention, as well as a *negligible memory footprint*. Since naive k-MIP attention already offers a speedup of a factor 50 over full attention in a training setting and of a factor 5 in an inference setting, the total speedup of k-MIP attention with symbolic matrices is a *factor 500 over full attention during training* and a *factor 50 during inference*, for large N.

In this section, we have evaluated the efficiency of three attention mechanisms: full self-attention, k-MIP attention with naive brute-force k-MIPS, and k-MIP attention with brute-force k-MIPS using symbolic matrices. We have demonstrated that k-MIP attention with symbolic matrices yields a speedup of roughly two orders of magnitude over full attention during both training and inference, while having a negligible memory footprint. This is enough to feasibly scale to graphs with up to 500K nodes, which we will do in section 6.6.

6.3 Negative Result: Ball Tree Search

While symbolic matrices greatly improve the efficiency of the brute-force k-MIP search, the computational complexity of this method remains quadratic in the number of nodes in the graph, which hinders its scalability to graphs with more than a million nodes. Therefore, we attempted to improve the computational complexity of the k-MIP search in the attention mechanism. Inspired by the promising results in [96], we opted to solve this problem using a Ball Tree Search (BTS) algorithm (explained in subsection 4.2.3) and implemented a PyTorch C++ extension for this. However, we found that the practical efficiency of this approach is not competitive with the naive GPU-accelerated brute-force k-MIPS, and therefore decided not to include it in the final model. This section explains why the BTS algorithm was not competitive, and what we have learnt from this negative result.

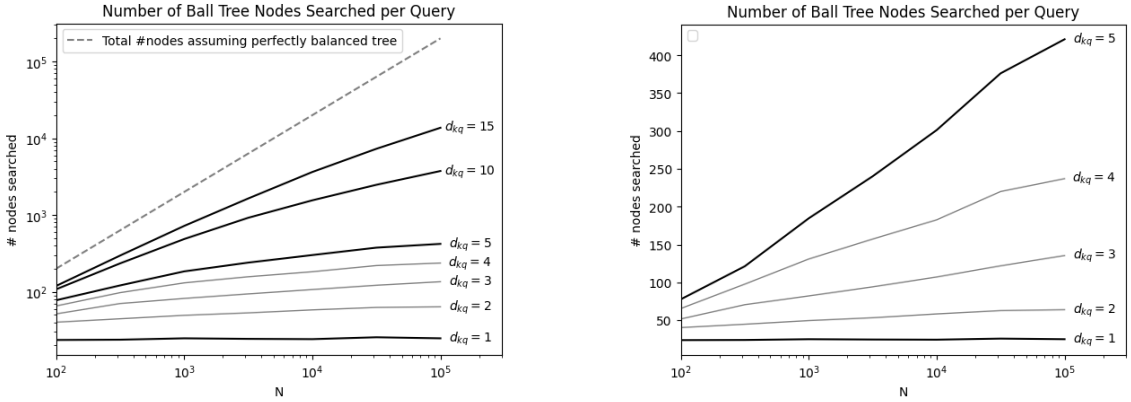
There are two main reasons why the BTS approach was not competitive. First, the

algorithm suffers from the curse of dimensionality: with each added dimension of the key-query space, the search space grows exponentially, making it increasingly difficult to efficiently partition and search the data. Second, the algorithm cannot be parallelized effectively on a GPU, which is a significant disadvantage compared to the brute-force approach. We don't exclude the possibility that it could be modified to run with hardware acceleration, but this would require a significant amount of work and is beyond the scope of this project.

In what follows, we will first demonstrate how the BTS algorithm suffers from the curse of dimensionality, and then compare its efficiency to the brute-force search with symbolic matrices.

6.3.1 Curse of Dimensionality

In this subsection, we conduct an experiment to demonstrate how the Ball Tree Search algorithm suffers from the curse of dimensionality. We will plot the number of nodes visited during the search as a function of the number of keys and queries N, for multiple values of the dimensionality of the key-query space d_{kq}. Then, we will compare this with the theoretical complexity of $\mathcal{O}(\log N)$.



(a) A log-log plot for $1 \leq d_{kq} \leq 15$.

(b) A linear-log plot, showing only $1 \leq d_{kq} \leq 5$.

Figure 6.3: Plots of the number of nodes visited during a Ball Tree Search for various dimensionalities of the key-query space d_{kq}, as a function of the number of nodes N.

Set-up For each run of this experiment, we sample normally distributed key and query matrices of size $N \times d_{kq}$, for each combination among $N \in \{100 \cdot \sqrt{10}^{i} \mid i =$

$0, \ldots, 6\}$ and $d_{kq} \in \{1, 2, 3, 4, 5, 10, 15\}$. Here, N is the number of tokens and d_{kq} is the dimensionality of the key-query space. We then run the Ball Tree Search algorithm described in subsection 4.2.3, and record the average number of ball tree nodes visited for each query. We repeat this experiment 5 times and plot the means in Figure 6.3.

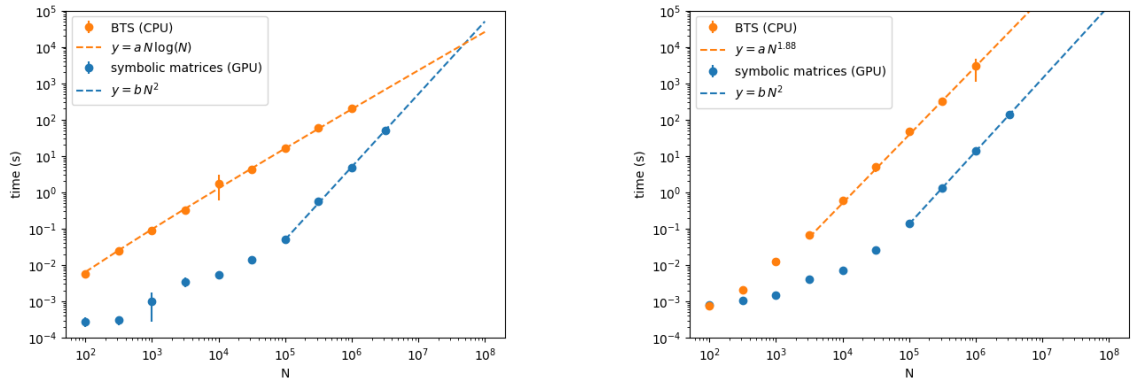
Results and Discussion The linear-log plot in Figure 6.3b shows that, for low-dimensional key-query spaces, the number of nodes visited during the search grows logarithmically with the dimensionality of the key-query space, which would lead to an overall complexity for the k-MIPS of $\mathcal{O}(N \log N)$. However, when the dimensionality is increased to larger values, the number of visited nodes quickly goes from logarithmic to linear growth w.r.t. the number of nodes. This is shown in the log-log plot in Figure 6.3a. When the complexity for each search is linear, the overall complexity of the k-MIP search becomes $\mathcal{O}(N^2)$.

This experiment demonstrates that the Ball Tree Search algorithm suffers from the curse of dimensionality, making the computational complexity of the k-MIP search higher than $\mathcal{O}(N \log N)$ when d_{kq} is large. This, in itself, is not that much of a problem, since we will demonstrate in section 6.5 that the k-MIP Graph Transformer is relatively robust to low values of d_{kq}. However, the following subsection demonstrates that the practical efficiency of our implementation of the BTS algorithm is not competitive with the brute-force approach, even for low-dimensional key-query spaces.

6.3.2 Comparison with Brute-Force Search

In this subsection, we compare in an experiment the efficiency of the Ball Tree Search algorithm to the brute-force search with symbolic matrices and GPU acceleration that we used in the other sections of this project.

Set-up For each run of this experiment, we sample normally distributed key and query matrices of size $N \times d_{kq}$, for $N \in \{100 \cdot \sqrt{10}^i \mid i = 0, \ldots, 6\}$ and $d_{kq} \in \{3, 10\}$. Here, N is the number of tokens and d_{kq} is the dimensionality of the key-query space.



(a) $d_{kq} = 3$ (b) $d_{kq} = 10$

Figure 6.4: Comparison of the efficiency of the Ball Tree Search algorithm and the brute-force search as a function of N, for $d_{kq} \in \{3, 10\}$.

We then run an attention pass with these keys and queries, using both the brute-force approach from subsection 4.2.2 and the Ball Tree Search algorithm described in subsection 4.2.3. We repeat this experiment 6 times, exclude the first "cold" run, record the time taken by both approaches, and plot the means in Figure 6.4. This experiment is run on an Intel® Core™ i7-9750H Processor, and the brute-force approach additionally has access to a NVIDIA GeForce RTX 2060. The time is measured in seconds.

Results and Discussion The results in Figure 6.4 show that the theoretical complexity that we derived in the previous section based on the number of visited nodes, is reflected in the practical efficiency of the Ball Tree Search algorithm. For $d_{kq} = 3$, the runtime of the BTS algorithm scales with $\mathcal{O}(N \log N)$, while for $d_{kq} = 10$, the runtime's asymptotic complexity is barely subquadratic. We fitted curves to the data points to illustrate this observation.

Further, we note the Ball Tree Search algorithm is significantly slower than the brute-force approach for both low-dimensional and high-dimensional key-query spaces. While the $\mathcal{O}(N \log N)$ complexity of BTS when d_{kq} is low will eventually outperform the $\mathcal{O}(N^2)$ complexity of the brute-force approach for very large N, the crossover point is so high that it is not worth it to use the BTS algorithm in practice. This is because, despite implementing the BTS algorithm in C++ and with the utmost care

for efficiency, it is hard to beat an algorithm that can run with hardware acceleration.

While we don't rule out the possibility that BTS could be modified to run with hardware acceleration, this would require a significant amount of work and is beyond the scope of this project. We will, however, try to implement this algorithm before preparing this dissertation for publication. For this, we will especially borrow from ideas from the field of computer graphics, where Bounding Volume Hierarchies (tree-based data structures used in ray tracing) with millions of triangles are generated and queried in a matter of milliseconds.

For now, because of its lower performance, we decided not to include the BTS algorithm in the final model and instead use the brute-force approach with symbolic matrices and GPU acceleration that was described in section 4.2.

6.4 Influence of k

The hyperparameter k, which determines the number of keys that each query attends to, has the potential to significantly influence the performance and runtime of the model. Understanding this dependency is crucial to make an informed choice for the hyperparameter k when scaling the model to larger datasets. In this section, we will therefore investigate the following research question:

Research question How do the efficiency and the performance of the k-MIP Graph Transformer vary with different values of k?

We hypothesize that, in terms of efficiency, the model's runtime will increase as k grows, since the softmax operation in the attention mechanism must be computed over k elements and k weighted value vectors must be aggregated. Regarding performance, we expect the model to improve with larger values of k, as it can take more context into account when processing each node. However, for larger values of k, there is a risk of overfitting if keys with low inner products introduce noise irrelevant to the query.

Set-up To study this research question, we train a k-MIP Graph Transformer on each of the datasets from section 6.1, but with varying values of k. The other hyperparameters of the models are summarized in section A.5. For each value of k, we conduct five runs and study the mean and standard deviation of both the test performance and the training epoch duration across these runs. The experiments are run on compute nodes with access to a single NVIDIA Tesla V100 GPU. The test performances are reported in Table 6.2 and the training epoch durations in Table 6.3. Further, we also create a scatter plot of the test performance against the average training epoch duration in Figure 6.5.

Table 6.2: Comparison of the performance of k-MIP-GT for varying k, on four small-graph datasets. Shown is the mean ± std over five runs. The **first** and **second** best results are highlighted.

k/Dataset	Cifar10 Accuracy ↑	MalNet-Tiny Accuracy ↑	PascalVOC-SP F1 score ↑	Peptides-Func AP ↑
$k = 1$	74.37 ± 0.56	93.00 ± 0.58	35.57 ± 1.06	60.64 ± 1.13
$k = 2$	74.84 ± 0.41	92.92 ± 0.69	**37.65 ± 0.70**	62.97 ± 0.27
$k = 3$	74.75 ± 0.33	92.88 ± 0.41	**37.47 ± 0.48**	63.13 ± 0.65
$k = 5$	75.08 ± 0.27	**93.54 ± 0.52**	37.30 ± 0.99	64.65 ± 0.37
$k = 10$	75.05 ± 0.32	**93.52 ± 0.49**	36.58 ± 1.50	65.10 ± 0.42
$k = 20$	75.12 ± 0.31	93.26 ± 0.45	35.80 ± 1.60	**65.31 ± 0.52**
$k = 30$	75.08 ± 0.48	92.62 ± 0.70	37.15 ± 0.73	65.14 ± 0.76
$k = 50$	**75.50 ± 0.43**	92.78 ± 0.75	36.82 ± 0.43	64.90 ± 0.84
$k = 100$	**75.65 ± 0.44**	93.40 ± 0.57	36.90 ± 0.33	**65.29 ± 0.34**

Table 6.3: Comparison of the average training epoch duration in seconds of k-MIP-GT for varying k, on four small-graph datasets. Shown is the mean ± std over five runs. The **first** and **second** best results are highlighted.

k/Dataset	Cifar10	MalNet-Tiny	PascalVOC-SP	Peptides-Func
$k = 1$	**135.34 ± 8.72**	**22.09 ± 1.19**	**16.14 ± 0.95**	**9.29 ± 0.29**
$k = 2$	146.64 ± 11.51	**23.96 ± 0.89**	**15.61 ± 1.00**	**9.70 ± 0.18**
$k = 3$	146.50 ± 20.20	24.25 ± 0.91	16.81 ± 1.11	10.05 ± 0.19
$k = 5$	147.94 ± 14.23	26.44 ± 0.90	17.12 ± 0.86	10.55 ± 0.25
$k = 10$	146.60 ± 19.97	30.14 ± 0.94	17.70 ± 0.89	12.79 ± 0.26
$k = 20$	**142.29 ± 20.11**	40.93 ± 1.15	22.58 ± 0.62	16.60 ± 0.38
$k = 30$	149.70 ± 19.76	51.39 ± 1.37	26.92 ± 1.35	21.40 ± 0.45
$k = 50$	166.97 ± 13.62	128.19 ± 2.27	73.42 ± 2.05	38.83 ± 0.84
$k = 100$	241.80 ± 19.67	666.36 ± 9.46	318.91 ± 5.11	150.91 ± 3.58

Results and Discussion It is clear from the results in Table 6.3 that the runtime of the model increases with k, as our intuition suggested. This increase is slow for

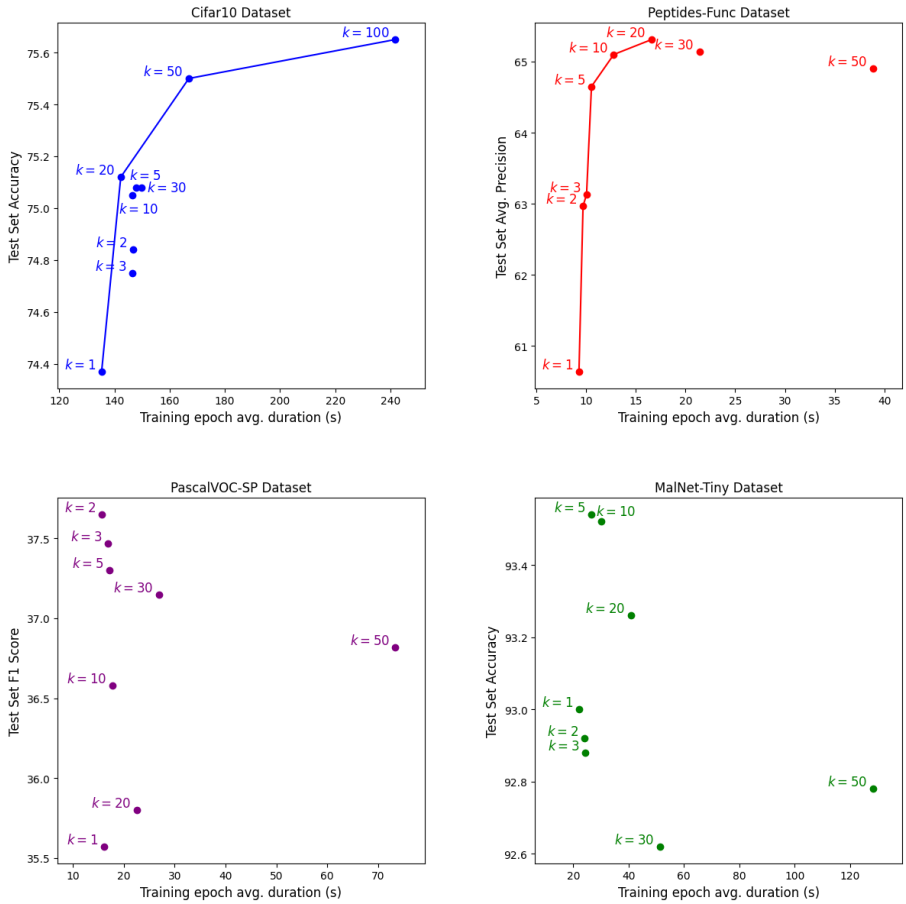


Figure 6.5: Scatter plots of the test performance against the average training epoch duration for varying values of k, for the datasets Cifar10, MalNet-Tiny, PascalVOC-SP, and Peptides-Func. The Pareto frontier is drawn for Cifar10 and Peptides-Func.

small values of k, when the attention mechanism is only a small part of the total computation time, but becomes more pronounced for larger values of k.

The performance of the model, on the other hand, is relatively stable for varying values of k. Only for the Cifar10 and the Peptides-Func datasets is there a clear signal that very low values of k are detrimental for the model's performance. For the other two datasets, the results are too noisy to draw a clear conclusion.

For Cifar10 and Peptides-Func, this leads to k presenting a trade-off between performance and efficiency, where a higher k leads to better performance but also longer training times. This trade-off is visualised in Figure 6.5, where we drew the Pareto frontier for Cifar10 and Peptides-Func. A value of $k = 10$ seems to be a good compromise for all four datasets, so we will use this value in the subsequent experiments and when scaling the model to larger datasets in section 6.6.

6.5 Influence of d_{kq}

When scaling to larger graphs, the k-MIP search quickly becomes the bottleneck of the k-MIP-GT. This search requires the computation of a large number of inner products, where each inner product requires a number of floating point operations proportional to the dimensionality of the key-query space d_{kq}. Further, section 6.3 discussed how non-brute-force methods for k-MIP search such as Ball Tree Search suffer from the curse of dimensionality, and thus also need to compute fewer inner products when the dimensionality of the key-query space d_{kq} is low. Therefore, it is of interest to investigate the following research question:

Research question How low can we make the dimensionality of the key-query space d_{kq}, while retaining good performance for the k-MIP Graph Transformer?

Set-up To answer this question, we will train a k-MIP Graph Transformer on each of the datasets from section 6.1, but with varying values of d_{kq}. The hyperparameters of the models are summarized in section A.6. We conduct five runs for each value of d_{kq}, and study the mean and standard deviation of the test performance across

these runs. The results of this experiment are shown in Table 6.4.

Table 6.4: Comparison of the performance of k-MIP-GT for varying d_{kq}, on four small-graph datasets. Shown is the mean ± std over five runs. The **first** and **second** best results are highlighted.

d_{kq}/**Dataset**	**Cifar10 Accuracy ↑**	**MalNet-Tiny Accuracy ↑**	**PascalVOC-SP F1 score ↑**	**Peptides-Func AP ↑**
$d_{kq} = 3$	75.37 ± 0.70	92.88 ± 0.58	37.09 ± 0.45	65.53 ± 1.04
$d_{kq} = 5$	**75.40 ± 0.38**	93.10 ± 0.57	**38.32 ± 1.34**	65.45 ± 0.73
$d_{kq} = 7$	**75.77 ± 0.27**	93.00 ± 0.63	**38.42 ± 0.92**	65.00 ± 0.22
$d_{kq} = 10$	75.15 ± 0.24	93.44 ± 0.72	37.62 ± 1.32	**65.78 ± 0.69**
$d_{kq} = 15$	75.26 ± 0.38	93.42 ± 0.81	38.02 ± 1.28	65.59 ± 0.14
$d_{kq} = 20$	75.06 ± 0.16	**93.54 ± 0.45**	37.66 ± 0.48	**65.82 ± 0.76**
$d_{kq} = 30$	74.26 ± 0.30	**93.60 ± 0.48**	38.09 ± 0.17	64.97 ± 1.23

Results and Discussion The results show that the performance of the k-MIP Graph Transformer is relatively stable for varying values of d_{kq}. Even if there is a slight decrease in performance for very low values of d_{kq}, it remains within the margin of error. Only for the MalNet-Tiny dataset is there a clear signal that a higher value of d_{kq} is beneficial for the model's performance. Comparing this result to the literature on sequence Transformers, our method shows significantly greater robustness to low values of d_{kq} than what has been observed in natural language processing [17]. We conjecture that determining compatibility between queries and keys in graph Transformers is either simpler or less critical than in natural language, where more rigid semantic dependencies between tokens impose stricter demands on the attention mechanism.

We conclude that the k-MIP Graph Transformer is relatively robust to the choice of d_{kq}, and that it is possible to use low values of d_{kq} to improve the efficiency of the k-MIP search without sacrificing much performance. In the next section, we will use $d_{kq} = 5$ when we scale our approach to larger datasets.

6.6 Scaling to Datasets with Larger Graphs

In this section, we present an empirical evaluation of the k-MIP Graph Transformer across four datasets of increasing size. We demonstrate that its performance consis-

tently ranks among the best of all evaluated baselines.

Datasets The datasets used in this study are Amazon-Computer [105], ShapeNet-Part [106], S3DIS [107], and ModelNet10 [108]. Detailed descriptions of these datasets can be found in section A.1. Although the latter three datasets are originally point cloud datasets, we converted them into graph datasets by constructing directed k-NN graphs from the point clouds, in line with previous work [109, 110, 111, 112]. The number of nearest neighbours for the k-NN graphs was set based on the optimal values reported in [109], with 8 nearest neighbours for ShapeNet-Part and ModelNet10, and 32 nearest neighbours for S3DIS.

These datasets vary in size: Amazon-Computer is a transductive node classification dataset containing a single graph of 14K nodes, ShapeNet-Part has 16.9K graphs with 2.6K nodes on average, S3DIS includes 23.5K graphs with 4.1K nodes on average, and ModelNet10 contains 4.9K graphs with 9.5K nodes on average. Notably, the largest graph in ModelNet10 has over 500K nodes.

Our goal was to select datasets that require capturing long-range dependencies, as graph Transformer models are particularly effective in such scenarios and hence likely to be employed in practice for this purpose. However, we observed a gap in traditional inductive graph machine learning datasets that contain more than 1K nodes and require the capture of long-range relationships. To address this, we opted to convert point cloud datasets into graph datasets. The intuition is that they naturally require the capture of long-range dependencies, because the global context is instrumental for accurate classification of each separate point.

Baselines We compare the k-MIP Graph Transformer to the same baselines as used in section 6.1: four traditional graph neural network architectures (GCN [33], GINE [84], GAT [34], and GatedGCN [85]) and four graph Transformer models (Exphormer [21], GPS+Transformer [67], GPS+BigBird [52, 67], and GPS+Performer [46, 67]). We do not evaluate GPS+Transformer on ShapeNet-Part, S3DIS, or ModelNet10 due to its high computational cost.

Set-up For each dataset/model combination, we train three models with different random seeds. For each such run, we save the test performance at the epoch where the validation performance is best, with performance in both cases given by the metric of interest for that dataset. In Table 6.5, we report the mean and standard deviation of the test performance across the three runs.

Due to resource constraints, it was not possible to conduct an extensive hyperparameter search for each model across all datasets. Instead, we selected the hyperparameters based on those reported in the Exphormer paper [21], while satisfying a parameter budget of 100K parameters for ModelNet10 and 300K parameters for the other three datasets. A detailed list of the hyperparameters and the rationale behind their selection can be found in section A.7.

Table 6.5: Test performance of k-MIP-GT and baselines on four datasets with increasingly larger graphs. Shown is the mean ± std over three runs. The **first** and **second** best results are highlighted.

Model/Dataset	Amazon-Computer Accuracy ↑	ShapeNet-Part F1 score ↑	S3DIS mIoU ↑	ModelNet10 Accuracy ↑
GCN	88.04 ± 1.15	60.18 ± 0.04	39.98 ± 1.47	79.11 ± 0.18
GINE	40.38 ± 0.92	64.57 ± 0.35	44.16 ± 0.62	79.04 ± 0.36
GAT	79.16 ± 2.29	63.01 ± 0.17	44.24 ± 1.14	77.37 ± 0.81
GatedGCN	**91.79 ± 0.32**	76.20 ± 0.32	63.53 ± 1.40	38.00 ± 0.72
GPS + BigBird	90.09 ± 0.35	79.65 ± 0.98	67.36 ± 0.26	78.82 ± 1.96
GPS + Performer	90.45 ± 0.17	77.36 ± 1.23	60.83 ± 0.56	74.67 ± 0.91
GPS + Transformer	90.35 ± 0.39	–	–	–
Exphormer	**91.53 ± 0.40**	**82.86 ± 0.31**	**68.37 ± 0.23**	**80.53 ± 0.66**
k-MIP-GT	90.86 ± 0.34	**83.03 ± 0.11**	**67.69 ± 1.29**	**81.15 ± 0.14**

Results and Discussion Table 6.5 shows the mean and standard deviation of the test performance across three runs for each method/dataset combination. We make two observations based on these results.

First, the k-MIP Graph Transformer consistently ranks among the best three of all evaluated baselines across all datasets and achieves a first place on two out of four datasets. This is a strong result, as Exphormer and GraphGPS-based models were state-of-the-art as little as a year ago.

Second, the fact that the k-MIP-GT – itself an instance of the GraphGPS framework – outperforms GPS+BigBird and GPS+Performer across all datasets is particularly

noteworthy. It demonstrates the effectiveness of the k-MIP attention mechanism for learning on graphs.

Third, while the graph Transformer models comfortably outperform the traditional graph neural networks on the ShapeNet-Part and S3DIS datasets (both inductive node classification on point cloud graphs), the difference is not as large on the Amazon-Computer and ModelNet10 datasets (transductive node classification and inductive graph classification, respectively). This demonstrates that the latter two datasets can be solved effectively with local models, and thus that graph Transformer models may be overkill for those tasks.

The largest graphs we trained the k-MIP-GT on were those from the ModelNet10 dataset, which has over 500K nodes in its largest graphs. Ironically, the limiting factor for scaling our method further (without resorting to graph subsampling) was the large memory footprint of the MPNN Layer, which becomes an issue when the number of edges in the input graph is large. This can be addressed by implementing distributed training across multiple GPUs, which we intend to explore before preparing this dissertation for publication.

To conclude, in this section we have evaluated the performance of the k-MIP Graph Transformer on four datasets of increasing size. We have demonstrated that its performance is competitive with state-of-the-art graph Transformer models, and that we have been able to scale the method to datasets with graphs of up to 500K nodes.

6.7 An Approximation for Full Attention?

While it is tempting to think about the k-MIP attention mechanism as a way to approximate full attention, we examine in this section whether this is indeed the case. Concretely, we will investigate the following research question:

Research question Does k-MIP attention approximate the full attention mechanism? If so, can we quantify the quality of this approximation?

Set-up To study this research question, we set up the following experiment. First, we extract the queries, keys, and values that serve as inputs to the layers of a fully trained k-MIP Graph Transformer. Specifically, we use a model trained on the MalNet-Tiny dataset, and extract the queries, keys, and values that are generated in the forward pass of the first 50 graphs in the test split of this dataset with more than 1000 nodes. Second, for these queries, keys and values, we compare the output of the k-MIP attention mechanism to the output of a full attention mechanism applied on the same vectors, and measure the L2 distance between both outputs for varying values of k. We report the average and the standard deviation of the L2 distances over each head and over the 50 graphs, for each value of k.

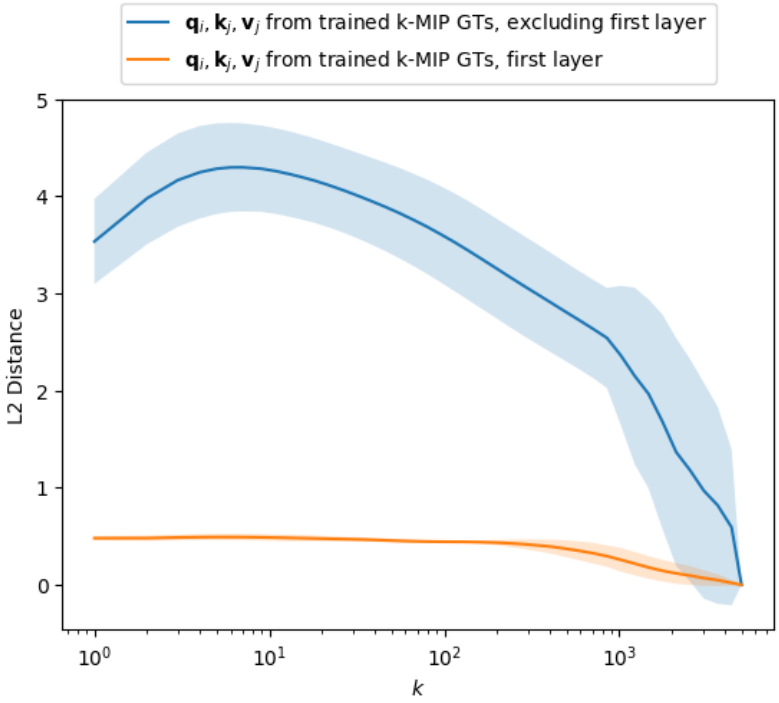


Figure 6.6: The L2 distance between the output of the k-MIP attention mechanism and the output of the full attention mechanism for varying k, where the inputs are queries, keys, and values extracted from a trained k-MIP Graph Transformer. The average and standard deviation are taken over each head and over the first 50 graphs in the test split of the MalNet-Tiny dataset. For reference, when approximating the full attention output with samples from $\mathcal{N}(\mathbf{0}_n, \boldsymbol{I}_n)$, the L2 distance was 4.5417 ± 0.3756.

Results and Discussion The results of this experiment are shown in Figure 6.6. We split our examination between the first layer of the model and the other layers,

because the behaviour of the first layer is quite different from the other layers. The figure shows that the approximation of the full attention mechanism with the k-MIP attention mechanism is very poor for all layers except the first, until k is close to the number of nodes in the graph. For reference: when approximating the full attention output with samples from a multivariate normal distribution, the L2 distance was 4.5417 ± 0.3756. The first layer, on the other hand, yields good approximations even for $k = 1$. Our hypothesis for this rests on the fact that a lot of the first layer's input tokens are identical. This results in a lot of the key-value pairs being identical as well, which leads to a good approximation for each query for which the maximum-inner-product key is a key that has many duplicates. This is not the case for the other layers, where the keys and values are more diverse and the approximation is worse.

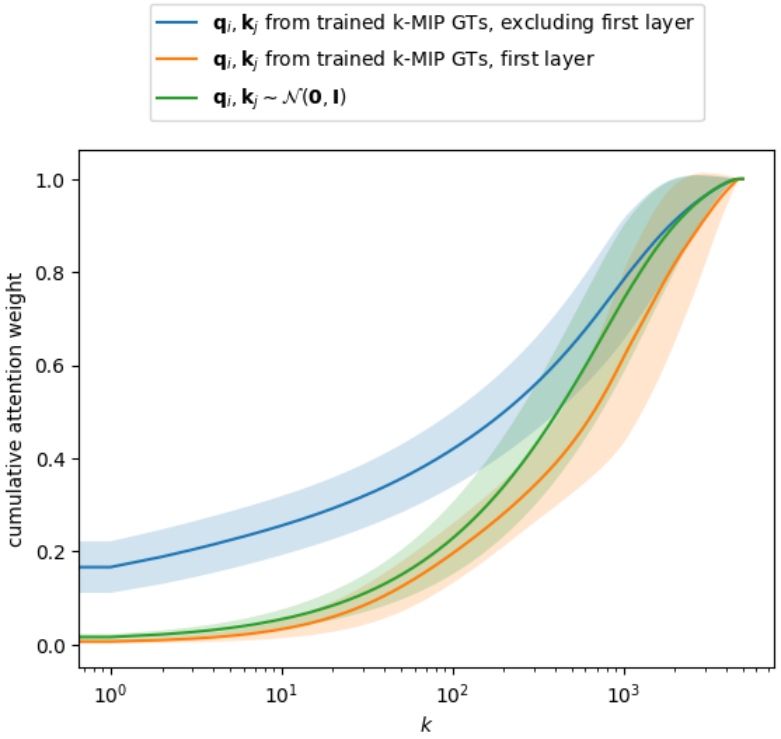


Figure 6.7: The cumulative attention weight for the k keys with the highest inner product for each query, for varying k, where the inputs are queries, keys, and values extracted from a trained k-MIP Graph Transformer. The average and the standard deviation are taken over each head and over the first 50 graphs in the test split of the MalNet-Tiny dataset. For reference, the same graph is plotted when the queries, keys, and values are sampled from a multivariate normal distribution.

The reason for the poor approximation of the full attention mechanism can be found by inspecting the attention weights. In Figure 6.7, we show the cumulative attention weight for the k keys with the highest inner product for each query, for various values of k. In the notation of section 4.1, we report the averages and standard deviations of

$$\sum_{i}^{N} \sum_{j \in K_i} \texttt{softmax}_{1 \leq j \leq N}(e_{ij}) \tag{6.1}$$

over each head and over the first 50 graphs in the test split of the MalNet-Tiny dataset.

The figure shows that, while the few highest-inner-product keys have a much larger attention weight than the other keys, they have only a small fraction of the total attention weight. It takes values of $k \approx 200$ to reach a cumulative attention weight of 0.5, and values of $k \approx 1000$ to reach a cumulative attention weight of 0.8. This implies that the many key-value pairs that do not belong to the k keys with the highest inner product can still have a significant total impact on the output of the attention mechanism, which cannot be taken into account by the k-MIP attention mechanism. This explains why the approximation is poor for values of k that are not close to the number of nodes in the graph. The first layer, on the other hand, shows reasonably low attention weights even for the few highest-inner-product keys. We also ascribe this to the fact that there are many identical key-value pairs in this layer's input: if the highest-inner-product key has many duplicates, each of these duplicates will have the same weight, resulting in a larger spread of the attention weights over the keys.

We conclude that the k-MIP attention mechanism does not approximate the full attention mechanism. Instead, it should be thought of as a different way to perform attention, which has different properties and can be more efficient than full attention.

6.8 Inspecting the attention graphs

In this section, we further investigate the k-MIP attention mechanism to gain more insight into the k-MIP Graph Transformer. We focus on the attention graphs that

are learnt by the model.

The model selects adaptively which k keys to attend to, for every query $\boldsymbol{q}_i$. In effect, this means that the model learns an attention graph on which to propagate information between nodes. In this attention graph G_{attn}, edges follow the flow of information, i.e. an edge exists from node j to node i if the model learns to attend to key $\boldsymbol{k}_j$ when processing query $\boldsymbol{q}_i$. Id est,

$$(j, i) \in E_{\text{attn}} \iff j \in K_i. \tag{6.2}$$

Further, the weight of the edge (j, i) is given by the attention score a_{ij}. We now study the following research question:

Research question How can we characterise the attention graphs learnt by the k-MIP Graph Transformer? What are the properties of these graphs, and how do they compare to the input graphs?

Set-up To answer this question, we will train a k-MIP Graph Transformer, and visualise the attention graphs learnt by the different heads of different layers of the model. Further, we will analyse the out-degree distributions in these graphs. For this experiment, we select the Cifar10 dataset because the input graphs are approximately planar (i.e. there are few edge crossings) and reasonably small, which makes them easier to visualise. The hyperparameters of the model and other experimental details are summarized in section A.9.

Results Figure 6.8 shows the first input graph in the test split of the Cifar10 dataset. By doing inference of the model on this graph and visualizing the resulting attention weights and attention graphs for each head and each layer, we get the results shown in Figure 6.9. In this figure, the edge alpha values are proportional to the corresponding attention weights and each node is coloured according to its out-degree for visualization purposes. In section B.1, we show analogous visualizations for the second, third and fourth input graphs in the test split of the Cifar10 dataset.

Discussion These visualizations allow us to gain insight into the structure of the attention graphs. We make three interesting observations. First, each head in each layer learns a significantly different attention graph. This suggests that each head learns to attend to different keys for each query, which is beneficial as it can enhance the diversity of information captured by the model, thereby contributing to its overall representational capacity. Second, the attention graphs show very little resemblance to the input graphs. This suggests that the model uses the attention mechanism to complement the information flow already present in the message-passing module of the GraphGPS framework. Third, in each head, there are some keys that are attended to by disproportionally many queries, which is reflected in the out-degree distributions of the nodes. Especially in the first layer of the model this is very pronounced.

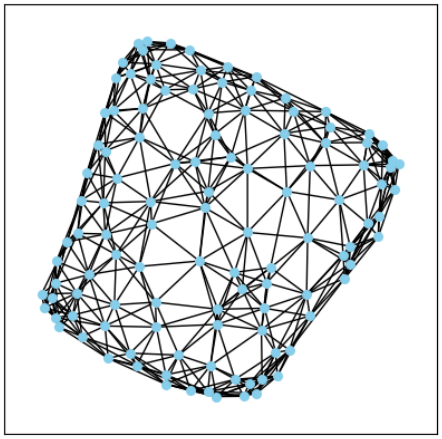


Figure 6.8: The first input graph in the test split of the Cifar10 dataset. The directionality of the edges was omitted for clarity, as most edges are bidirectional.

To further corroborate the third observation, we study the out-degree distribution in the learnt attention graphs. Figure 6.10 visualises the out-degree distributions of each layer, averaged over the first 200 graphs and all heads of that layer in the test split of the Cifar10 dataset. The left column shows a histogram of the occurrences of the out-degrees, whereas the right column shows a histogram of the total number of edges that depart from nodes with a certain out-degree. The means and standard deviations are taken over five models trained with different random seeds.

These figures confirm our previous observation that some keys get attended to by

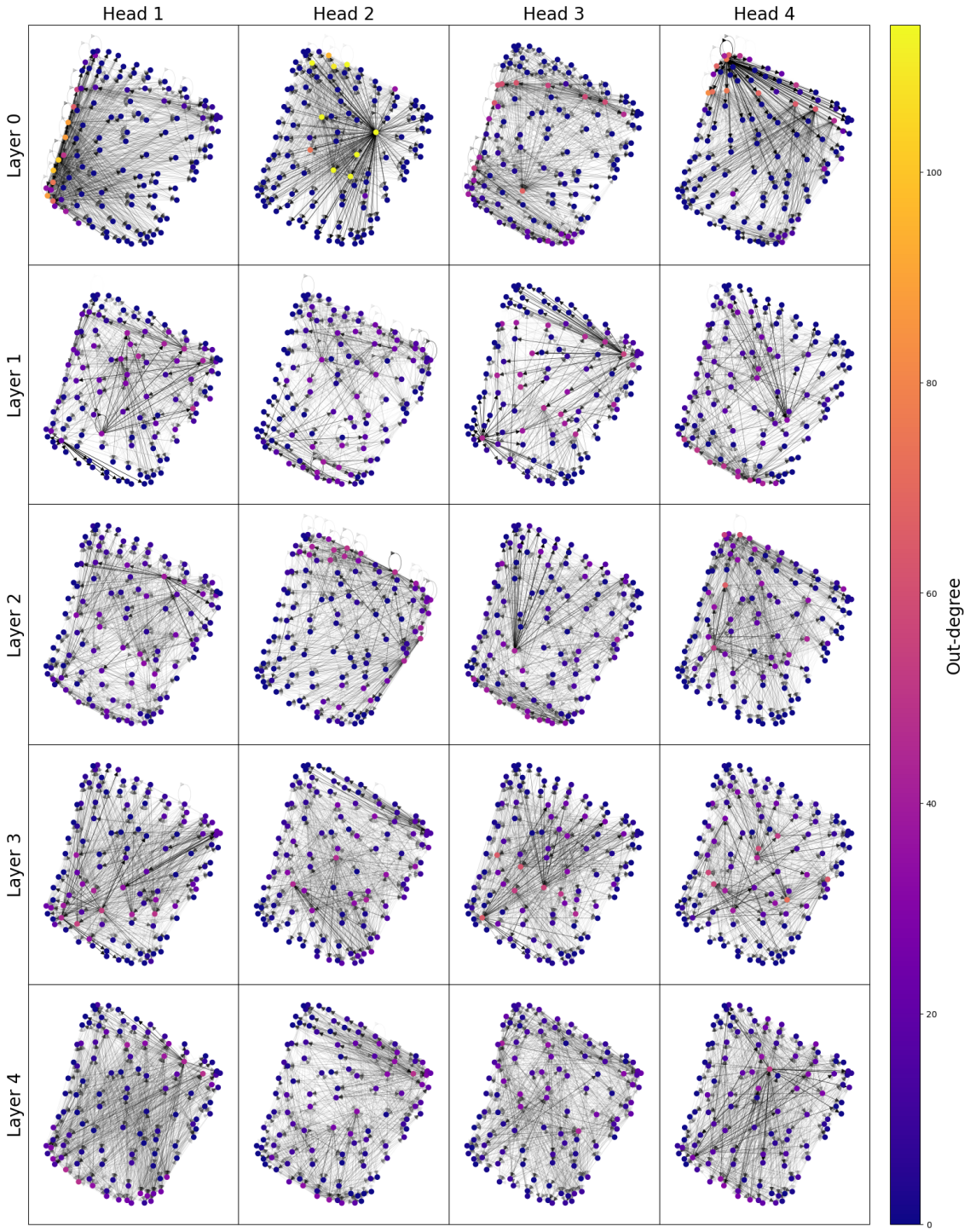


Figure 6.9: The learnt attention graphs of each head and each layer in a trained k-MIP-GT, for the first input graph in the test split of the Cifar10 dataset. The edge alpha values are proportional to the corresponding attention weights. Each node is coloured according to its out-degree.

disproportionally many queries, and especially in the first layer: the out-degrees follow a long-tailed distribution, where there are a few nodes with high out-degree from

which most of the edges depart. This long-tailed distribution is very pronounced in the first layer of the model, and becomes less prominent in the deeper layers.

Like in section 6.7, our hypothesis for this phenomenon rests on the fact that a lot of the first layer's input tokens are identical. Because duplicates will attend to the same k keys, this results in a few keys being attended to by many queries. This is not the case for the other layers, where the keys are more diverse and, while still showing a long-tailed distribution, the out-degrees are more evenly distributed.

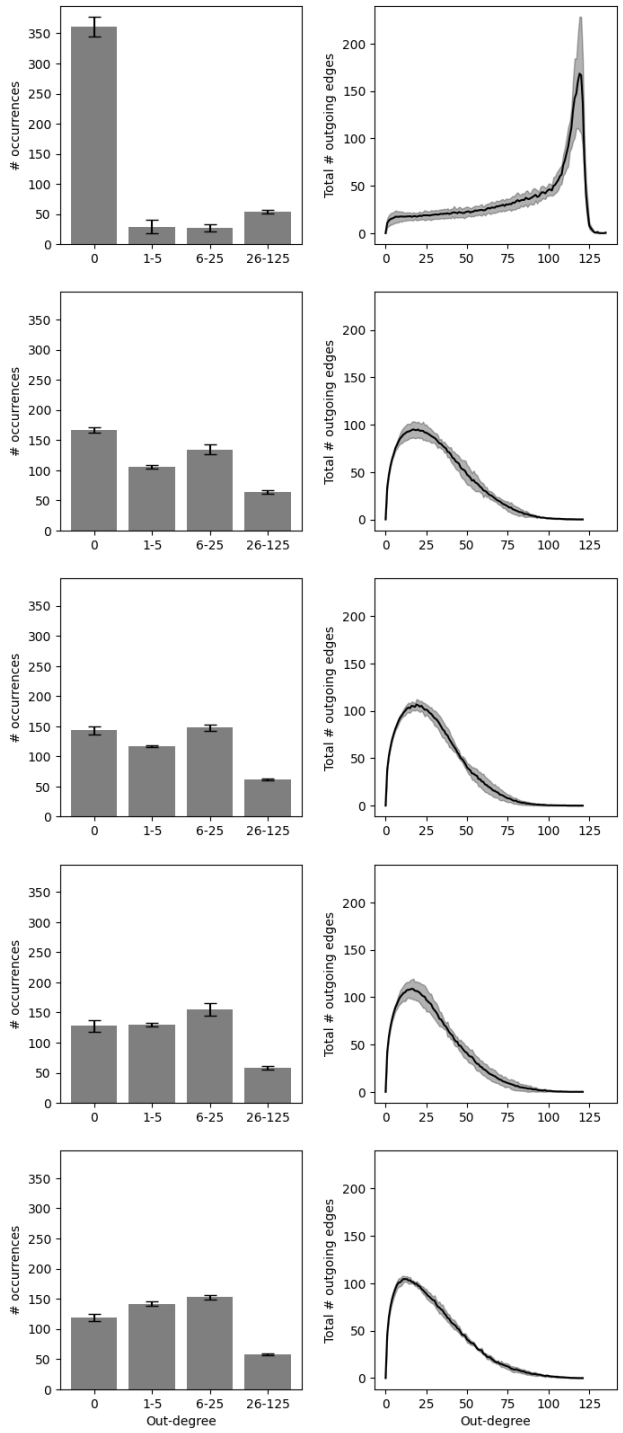


Figure 6.10: Statistics on the out-degree distributions in the attention graphs in k-MIP-GTs trained on Cifar10. Left: the distributions of the out-degrees, divided into 4 bins. Right: the contributions of nodes with a certain out-degree to the total number of edges. In both cases, the out-degrees are collected over the first 1000 graphs in the Cifar10 dataset and over all heads in the corresponding layer. Then, the mean $\pm$ std is plotted over five models trained on Cifar10 with different random seeds.

7 | Conclusion

In this dissertation, we aimed to develop a graph Transformer architecture that can scale efficiently to large graphs while overcoming the key limitations of existing methods. Specifically, many current scalable graph Transformers struggle with performance loss due to the linearisation of the attention mechanism, rigid attention patterns that limit flexibility, and the inability of graph subsampling methods to capture long-range dependencies.

To this end, we introduced the k-MIP Graph Transformer, a novel graph Transformer architecture that dynamically selects the most influential nodes to attend to using the k-MIP self-attention mechanism. We have demonstrated that the k-MIP Graph Transformer is:

- **Efficient:** The k-MIP self-attention mechanism is two orders of magnitude faster than full attention and has a negligible memory footprint, allowing us to scale to graphs with up to 500K nodes.
- **Versatile:** The k-MIP-GT incorporates edge features and supports node-level, graph-level, and edge-level tasks.
- **Performant:** We have demonstrated results competitive with prominent graph Transformers across a variety of graph learning tasks, with graphs ranging from 20 to 500K nodes.
- **Expressive:** We have established universal approximation guarantees for the k-MIP Graph Transformer, analogous to those previously established for full-attention Transformers and graph Transformers.

Additionally, our analysis has yielded valuable insights into the behaviour of the k-MIP-GT, highlighting that (1) the parameter k can offer a trade-off between performance and efficiency, (2) the k-MIP-GT is relatively robust to low-dimensional key-query spaces, and (3) the k-MIP-GT should not be thought of as an approximation to the full self-attention mechanism.

By introducing the k-MIP Graph Transformer, we have demonstrated that dynamically selecting which nodes to attend to is a viable and efficient alternative to full attention, offering scalability without compromising performance. We hope that this novel approach will encourage further exploration in this direction, sparking innovations in graph Transformer architectures and ultimately enabling the application of more powerful graph learning models to previously intractable tasks.

While we consider this dissertation a success, there remain several areas where further refinement could enhance the work before publication. In the rest of this chapter, we will discuss these opportunities for improvement, as well as potential avenues for future research.

7.1 Limitations and Opportunities

While the results presented in this dissertation demonstrate the effectiveness and efficiency of the k-MIP Graph Transformer, we identify several key aspects of the method that present opportunities for refinement. Some of these are inherent challenges, while others represent practical adjustments that can be made before sharing our work with the research community. We identify four of these aspects below.

Further scaling The largest graphs we trained our method on were those from the ModelNet10 dataset (>500K nodes in its largest graphs). Ironically, the limiting factor for scaling our method further (without resorting to graph subsampling) was the large memory footprint of the MPNN Layer, which becomes an issue when the number of edges in the input graph is large. To overcome this, we plan to implement distributed training across multiple GPUs, which will enable training on graphs in the order of millions of nodes, until the quadratic complexity of the k-MIP search becomes the limiting factor.

Hyperparameter tuning Due to resource constraints, we have not performed an extensive hyperparameter search and instead opted to start from the hyperparameters in the Exphormer repository [21]. While this yielded promising results,

we believe that further optimisation is possible for both the baseline methods and our own methods. Before publication, we plan to perform a more extensive hyper-parameter search for the Transformer methods and use state-of-the-art numbers for the baselines when available. This approach will ensure a maximal empirical rigor and a fair comparison between all methods.

Computational complexity While we achieved a significant speedup over the full attention mechanism and scaled our method to graphs with up to 500K nodes, the computational complexity of the k-MIP-GT remains quadratic in the number of nodes due to the k-MIPS operation. Before publication, we plan to address this.

Our Ball Tree Search method, despite poor practical performance, showed that a linearithmic complexity is possible. Hence, we believe that a hardware-aware implementation of a binary search algorithm could yield a solution that is both efficient and scalable. While the literature and open-source frameworks from the field of information retrieval did not yield competitive results in preliminary experiments, this may be due to the fact that they are optimised for search in datasets with billions of keys, where the construction time of the index is amortized. In further attempts to resolve this, we will borrow from ideas in computer graphics, where Bounding Volume Hierarchies (tree-based data structures used in ray tracing) with millions of triangles are generated and queried in a matter of milliseconds. We will search for an existing implementation that can be adapted to our needs, or develop our own if necessary.

Supervision starvation An inherent property of the k-MIP-GT is that the model gets to see only a subset of the keys for each query in each step. If the relevant keys for a query never reach the top k keys during training, the model will not be able to learn this attention pattern. This is in constrast to full attention, where there is a gradient flow from every intermediate token activation to each key and each query. We leave it for future work to investigate whether this is a significant issue in practice, and if so, to develop strategies to mitigate it. One possible solution could be to start with a large k and gradually decrease it as training progresses, allowing

the model to learn from a wider range of keys in the beginning and then focus on the most relevant ones as it converges.

7.2 Future Work

Building on the foundations established in this dissertation, we identify the following three avenues for future work.

Exploration of other key-query spaces In this dissertation, we embedded the keys and queries into a vector space with an inner product metric. While this choice was motivated by the success of standard dot-product attention, it would be interesting to see how other metric choices for the key-query space affect the performance and efficiency of the k-MIP-GT. A limited amount of work has been done in this direction for standard Transformers [113, 114] and graph Transformers [115, 116]. In the case of the k-MIP-GT, however, the choice of key-query space is even more crucial, as a space that can be more easily partitioned could lead to a more efficient search algorithm.

Encouraging diversity among attended keys In the k-MIP attention mechanism discussed in this dissertation, we have focused on attending to the k keys for each query that have the highest inner product with that query, in order to closely mirror the full attention mechanism. However, encouraging the model to attend to a more diverse set of keys may lead to more robust representations, particularly in settings where a wider range of attention patterns could capture critical information present in the data. Investigating how such diversity affects both model performance and interpretability could provide valuable insights for further refinement.

k-MIP attention with expander graphs We noted in our experiments that Exphormer [21] was very often among the best performing of all the baseline methods. Since their main innovation – the use of expander graphs – is orthogonal to our approach, it may be promising to investigate whether the performance of the k-MIP-GT could be enhanced by performing attention on expander graphs alongside

k-MIP attention.

7.3 Final Remark

By introducing the k-MIP Graph Transformer, we have demonstrated that dynamically selecting the most influential nodes to attend to is a viable and efficient alternative to full attention, offering scalability without compromising performance.

Once the limitations outlined in this chapter have been addressed, we intend to share our findings with the wider research community. It is our hope that this work will encourage further exploration into dynamic attention mechanisms, ultimately enabling the use of more powerful graph learning models for previously intractable tasks, such as large-scale weather forecasting and molecular modelling.

Bibliography

[1] F. Borisyuk, S. He, Y. Ouyang, M. Ramezani, P. Du, X. Hou, C. Jiang, N. Pasumarthy, P. Bannur, B. Tiwana *et al.*, "Lignn: Graph neural networks at linkedin," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 4793–4803.

[2] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[3] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 639–648.

[4] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.

[5] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals," *Chemistry of Materials*, vol. 31, no. 9, pp. 3564–3572, 2019.

[6] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Physical review letters*, vol. 120, no. 14, p. 145301, 2018.

[7] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[8] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *Advances in neural information processing systems*, vol. 30, 2017.

[9] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu *et al.*, "Graphcast: Learning skillful medium-range global weather forecasting," *arXiv preprint arXiv:2212.12794*, 2022.

[10] R. Keisler, "Forecasting global weather with graph neural networks," *arXiv preprint arXiv:2202.07575*, 2022.

[11] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[12] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," *arXiv preprint arXiv:2006.05205*, 2020.

[13] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[15] A. Loukas, "What graph neural networks cannot learn: depth vs width," *arXiv preprint arXiv:1907.03199*, 2019.

[16] S. Akansha, "Over-squashing in graph neural networks: A comprehensive survey," *arXiv preprint arXiv:2308.15568*, 2023.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 387–27 401, 2022.

[19] Q. Wu, C. Yang, W. Zhao, Y. He, D. Wipf, and J. Yan, "Difformer: Scalable (graph) transformers induced by energy constrained diffusion," *arXiv preprint arXiv:2301.09474*, 2023.

[20] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, "Simplifying and empowering transformers for large-graph representations," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[21] H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop, "Exphormer: Sparse transformers for graphs," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 613–31 632.

[22] A. Alboody and R. Slama, "Graph transformer mixture-of-experts (gtmoe) for 3d hand gesture recognition," in *Intelligent Systems Conference*. Springer, 2024, pp. 317–336.

[23] C. M. Frey, Y. Ma, and M. Schubert, "Sea: graph shell attention in graph neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2022, pp. 326–343.

[24] Z. Qin, W. Sun, H. Deng, D. Li, Y. Wei, B. Lv, J. Yan, L. Kong, and Y. Zhong, "cosformer: Rethinking softmax in attention," *arXiv preprint arXiv:2202.08791*, 2022.

[25] P. Wang, X. Wang, F. Wang, M. Lin, S. Chang, H. Li, and R. Jin, "Kvt: k-nn attention for boosting vision transformers," in *European conference on computer vision*. Springer, 2022, pp. 285–302.

[26] G. Zhao, J. Lin, Z. Zhang, X. Ren, Q. Su, and X. Sun, "Explicit sparse transformer: Concentrated attention through explicit selection," *arXiv preprint arXiv:1912.11637*, 2019.

[27] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar, "Are transformers universal approximators of sequence-to-sequence functions?" *arXiv preprint arXiv:1912.10077*, 2019.

[28] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 618–21 629, 2021.

[29] O. Wieder, S. Kohlbacher, M. Kuenemann, A. Garon, P. Ducrot, T. Seidel, and T. Langer, "A compact review of molecular property prediction with graph neural networks," *Drug Discovery Today: Technologies*, vol. 37, pp. 1–12, 2020.

[30] A. Sankar, Y. Liu, J. Yu, and N. Shah, "Graph neural networks for friend ranking in large-scale social platforms," in *Proceedings of the Web Conference 2021*, 2021, pp. 2535–2546.

[31] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE international conference on data mining (ICDM)*. IEEE, 2019, pp. 598–607.

[32] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[34] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48 550, 2017.

[35] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.

[36] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *arXiv preprint arXiv:2111.14522*, 2021.

[37] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in neural information processing systems*, vol. 34, pp. 28 877–28 888, 2021.

[38] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2020.

[40] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 6706–6713.

[41] S. R. Choi and M. Lee, "Transformer architecture and attention mechanisms in genome data analysis: a comprehensive review," *Biology*, vol. 12, no. 7, p. 1033, 2023.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986," *Biometrika*, vol. 71, no. 599-607, p. 6, 1986.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[44] T. Yi, D. Mostafa, B. Dara, and M. Donald, "Efficient transformers: A survey," *ACM Computing Surveys*, 2022.

[45] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[46] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=Ua6zuk0WRH

[47] F. X. X. Yu, A. T. Suresh, K. M. Choromanski, D. N. Holtmann-Rice, and S. Kumar, "Orthogonal random features," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/53adaf494dc89ef7196d73636eb2451b-Paper.pdf

[48] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*. PMLR, 2020, pp. 5156–5165.

[49] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[50] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[51] J. Ainslie, S. Ontanon, C. Alberti, V. Cvicek, Z. Fisher, P. Pham, A. Ravula, S. Sanghai, Q. Wang, and L. Yang, "Etc: Encoding long and structured inputs in transformers," *arXiv preprint arXiv:2004.08483*, 2020.

[52] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[53] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[54] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[55] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569.

[56] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[57] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, 2021.

[58] S. Wang, L. Zhou, Z. Gan, Y.-C. Chen, Y. Fang, S. Sun, Y. Cheng, and J. Liu, "Cluster-former: Clustering-based sparse transformer for long-range dependency encoding," *arXiv preprint arXiv:2009.06097*, 2020.

[59] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, "Sparse sinkhorn attention," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9438–9447.

[60] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast transformers with clustered attention," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 665–21 674, 2020.

[61] H. S. de Oc'ariz Borde, "Elucidating graph neural networks, transformers, and graph transformers," Unpublished manuscript retrieved from ResearchGate, 2024. [Online]. Available: https://www.researchgate.net/profile/Haitz_Saez_De_Ocariz_Borde/publication/378394991_Elucidating_Graph_Neural_Networks_Transformers_and_Graph_Transformers/links/65d793e2c3b52a1170eacabe/Elucidating-Graph-Neural-Networks-Transformers-and-Graph-Transformers.pdf

[62] J. Kim, D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong, "Pure transformers are powerful graph learners," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 582–14 595, 2022.

[63] S. Luo, T. Chen, Y. Xu, S. Zheng, T.-Y. Liu, L. Wang, and D. He, "One transformer can understand both 2d & 3d molecular data," in *The Eleventh International Conference on Learning Representations*, 2022.

[64] W. Park, W. Chang, D. Lee, J. Kim, and S.-w. Hwang, "Grpe: Relative positional encoding for graph transformer," *arXiv preprint arXiv:2201.12787*, 2022.

[65] W. Zhu, T. Wen, G. Song, L. Wang, and B. Zheng, "On structural expressive power of graph transformers," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 3628–3637.

[66] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," *arXiv preprint arXiv:2011.07682*, 2020.

[67] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 501–14 515, 2022.

[68] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[69] K. Kong, J. Chen, J. Kirchenbauer, R. Ni, C. B. Bruss, and T. Goldstein, "Goat: A global transformer on large-scale graphs," in *International Conference on Machine Learning*. PMLR, 2023, pp. 17 375–17 390.

[70] J. Park, S. Yun, H. Park, J. Kang, J. Jeong, K.-M. Kim, J.-W. Ha, and H. J. Kim, "Deformable graph transformer," *arXiv preprint arXiv:2206.14337*, 2022.

[71] J. Zhao, C. Li, Q. Wen, Y. Wang, Y. Liu, H. Sun, X. Xie, and Y. Ye, "Gophormer: Ego-graph transformer for node classification," *arXiv preprint arXiv:2110.13094*, 2021.

[72] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[73] J. Chen, K. Gao, G. Li, and K. He, "Nagphormer: A tokenized graph transformer for node classification in large graphs," *arXiv preprint arXiv:2206.04910*, 2022.

[74] D. Masters, J. Dean, K. Klaser, Z. Li, S. Maddrell-Mander, A. Sanders, H. Helal, D. Beker, L. Rampášek, and D. Beaini, "Gps++: An optimised hybrid mpnn/transformer for molecular property prediction," *arXiv preprint arXiv:2212.02229*, 2022.

[75] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[76] R. Sato, M. Yamada, and H. Kashima, "Random features strengthen graph neural networks," in *Proceedings of the 2021 SIAM international conference on data mining (SDM)*. SIAM, 2021, pp. 333–341.

[77] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," *arXiv preprint arXiv:2110.07875*, 2021.

[78] D. Lim, J. Robinson, L. Zhao, T. Smidt, S. Sra, H. Maron, and S. Jegelka, "Sign and basis invariant networks for spectral graph representation learning," *arXiv preprint arXiv:2202.13013*, 2022.

[79] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4465–4478, 2020.

[80] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE Trans-*

actions on Pattern Analysis and Machine Intelligence, vol. 45, no. 1, pp. 657–668, 2022.

[81] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3469–3489.

[82] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.

[83] M. Petrache and S. Trivedi, "Approximation-generalization trade-offs under (approximate) group equivariance," *Advances in Neural Information Processing Systems*, vol. 36, pp. 61 936–61 959, 2023.

[84] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," *arXiv preprint arXiv:1905.12265*, 2019.

[85] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[86] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 43, pp. 1–48, 2023.

[87] W. L. Hamilton, *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[88] L. Zehui, P. Liu, L. Huang, J. Chen, X. Qiu, and X. Huang, "Dropattention: A regularization method for fully-connected self-attention networks," *arXiv preprint arXiv:1907.11065*, 2019.

[89] G. Rattan and T. Seppelt, "Weisfeiler-leman and graph spectra," in *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2023, pp. 2268–2285.

[90] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.

[91] Y. Guo, D. Stutz, and B. Schiele, "Robustifying token attention for vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 557–17 568.

[92] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[93] E. Bernhardsson, *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018, python package version 1.13.0. [Online]. Available: https://pypi.org/project/annoy/

[94] "Torchpq," https://github.com/facebookresearch/TorchPQ, 2021.

[95] J. Feydy, J. Glaunès, B. Charlier, and M. Bronstein, "Fast geometric learning with symbolic matrices," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[96] P. Ram and A. G. Gray, "Maximum inner-product search using tree data-structures," *arXiv preprint arXiv:1202.6101*, 2012.

[97] P. Nithyasri, "Kd-tree and ball tree algorithms," https://medium.com/@polkamnithyasri/kd-tree-and-ball-tree-algorithms-aa99dfad4ceb, accessed: 2024-08-25.

[98] Python Software Foundation, "Python language reference." [Online]. Available: https://www.python.org/

[99] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[100] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[101] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[102] L. Biewald, "Weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

[103] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[104] J. Tönshoff, M. Ritzert, E. Rosenbluth, and M. Grohe, "Where did the gap go? reassessing the long-range graph benchmark," *arXiv preprint arXiv:2309.00367*, 2023.

[105] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," 2019. [Online]. Available: https://arxiv.org/abs/1811.05868

[106] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3d shape collections," *ACM Transactions on Graphics (ToG)*, vol. 35, no. 6, pp. 1–12, 2016.

[107] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1534–1543.

[108] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of*

the IEEE conference on computer vision and pattern recognition, 2015, pp. 1912–1920.

[109] Q. Zheng, Y. Qi, C. Wang, C. Zhang, and J. Sun, "Pointvig: A lightweight gnn-based model for efficient point cloud analysis," *arXiv preprint arXiv:2407.00921*, 2024.

[110] Z. Liang, M. Yang, L. Deng, C. Wang, and B. Wang, "Hierarchical depthwise graph convolutional neural network for 3d semantic segmentation of point clouds," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8152–8158.

[111] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4548–4557.

[112] C. Wang, B. Samari, and K. Siddiqi, "Local spectral graph convolution for point set feature learning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 52–66.

[113] C. Gulcehre, M. Denil, M. Malinowski, A. Razavi, R. Pascanu, K. M. Hermann, P. Battaglia, V. Bapst, D. Raposo, A. Santoro *et al.*, "Hyperbolic attention networks," *arXiv preprint arXiv:1805.09786*, 2018.

[114] D. Konstantinidis, I. Papastratis, K. Dimitropoulos, and P. Daras, "Multi-manifold attention for vision transformers," *IEEE Access*, 2023.

[115] S. Cho, S. Cho, S. Park, H. Lee, H. Lee, and M. Lee, "Curve your attention: Mixed-curvature transformers for graph representation learning," *arXiv preprint arXiv:2309.04082*, 2023.

[116] M. Yang, H. Verma, D. C. Zhang, J. Liu, I. King, and R. Ying, "Hypformer: Exploring efficient transformer fully in hyperbolic space," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '24. New York, NY, USA:

Association for Computing Machinery, 2024, p. 3770–3781. [Online]. Available: https://doi.org/10.1145/3637528.3672039

[117] V. P. Dwivedi, L. Rampášek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini, "Long range graph benchmark," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 326–22 340, 2022.

[118] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, 2015, pp. 43–52.

[119] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

A | Experimental Details

A.1 Datasets

Table A.1: Overview of the graph learning datasets used in this study.

Dataset	# Graphs	Avg. # nodes	Avg. # edges	Directed	Prediction level	Prediction task	Metric
CIFAR10	60,000	117.6	941.1	Yes	graph	10-class classif.	Accuracy
MalNet-Tiny	5,000	1,410.3	2,859.9	Yes	graph	5-class classif.	Accuracy
PascalVOC-SP	11,355	479.4	2,710.5	No	ind. node	21-class classif.	F1 score
Peptides-Func	15,535	150.9	307.3	No	graph	10-task classif.	Avg. Prec.
Amazon-Computer	1	13,752	491,722	No	transd. node	10-class classif.	Accuracy
ShapeNet-Part	16,881	2,616.2	20,929.6	Yes	ind. node	50-class classif.	F1 score
S3DIS	23,585	4,096.0	131,072.0	Yes	ind. node	12-class classif.	mIoU
ModelNet10	4,899	9,508.2	76,065.6	Yes	graph	10-class classif.	Accuracy

CIFAR10 The Cifar10 graph dataset [86] was derived from the homonymous image classification dataset by constructing an 8-NN graph on SLIC superpixels extracted from the images. The node features are 5-dimensional and consist of 3 RGB values and (x,y)-coordinates for the superpixel. The edge features are 1-dimensional and consist of the node distance. The task is to classify the images into one of ten classes. The splits are the same as in the original dataset: 45K/5K/5K for training/validation/test respectively.

MalNet-Tiny MalNet-Tiny [67] is a subset of MalNet [66] that is comprised of function call graphs derived from Android APKs. This subset contains 5,000 graphs with of up to 5000 nodes, each coming from benign software or 4 types of malware. The graphs are stripped of any original node or edge features, the task is to predict the type of the software based on the structure alone. [67].

PascalVOC-SP The PascalVOC-SP dataset is part of the Long Range Graph Benchmark (LRGB) [117]. It is derived from the Pascal VOC 2011 image classification dataset by constructing adjacency (hence planar) graphs on SLIC superpixels extracted from the images. The task is to classify each superpixel into one of 21 semantic segmentation classes.

Peptides-func The Peptides-func dataset is also part of the Long Range Graph Benchmark (LRGB) [117]. It consists of atomic graphs of peptides obtained from the SATPdb dataset. The task is multi-label graph classification, where each peptide is classified into one or more of 10 functional classes.

Amazon-Computer Amazon-Computer [105] is a segment of the Amazon co-purchase graph [118]. Each node represents a products in the "Computers" category, edges indicate that two goods are frequently bought together, and node features are bag-of-word encoded product reviews. The class label are given by the product category.

ShapeNet-Part The ShapeNet-Part dataset [106] is a subset of the ShapeNet repository [119], designed for 3D part segmentation tasks. It contains 16,881 3D shapes from 16 object categories. Each shape is annotated with 2-6 parts, with a total of 50 distinct part labels across all categories. The dataset is provided as 3D point clouds with corresponding part labels for each point. We transformed this point cloud segmentation task into a node classification task by constructing a directed k-NN graph on the point cloud where $k = 8$, which is the optimal k found for ModelNet40 [108] in [109]. We use the train/test/validation split from PyTorch Geometric [100].

S3DIS The Stanford Large-Scale 3D Indoor Spaces (S3DIS) dataset [107] consists of the 3D point clouds of six large-scale indoor areas from three different buildings of Stanford university. The point features are the 3D positions and RGB values, and each point is labelled as one of 13 semantic classes. We transformed this point cloud segmentation task into a node classification task by constructing a directed k-NN graph on the point cloud where $k = 32$, which is the optimal k found for this dataset in [109]. We use the train/test split from PyTorch Geometric [100], and randomly select 3,294 graphs from the training set for validation. This results in 16,997 training graphs, 3,294 validation graphs, and 3,294 test graphs.

ModelNet10 The ModelNet10 dataset [108] is a part of ModelNet40 dataset [108], consisting of 4,899 pre-aligned CAD models from 10 distinct object categories. The used point features are the surface normals and the 3D coordinates of the points. We do not use the provided triangles. We transformed this point cloud classification task into a graph classification task by constructing a directed k-NN graph on the point cloud where $k = 8$, which is the optimal k found for ModelNet40 in [109]. We use the train/test split from PyTorch Geometric [100] and randomly select 25% of the training set for validation. This results in 3,083 training graphs, 908 validation graphs, and 908 test graphs.

A.2 Hyperparameters: Performance on Various Small-Graph Datasets

Table A.2 displays the hyperparameters used in the Transformer-based models of the experiment in section 6.1. These hyperparameters are exactly the configuration of the Exphormer model from the Exphormer paper [21] and this configuration was used for all Transformer-based models, except from three minor modifications. First, the batch size for GPS+Transformer was lowered for the MalNet-Tiny dataset to avoid running out of memory. Second, we used 4 heads instead of 8 for the PascalVOC-SP dataset, as this led to better performance almost across the board and including for Exphormer itself. Third, the hidden dimension of each model was adapted to fit a fixed parameter budget. The chosen parameter budget is 100K for Cifar10, 300K for MalNet-Tiny and 500K for PascalVOC-SP and Peptides-Func. The hidden dimension, as well as the resulting number of parameters for each model is summarized in Table A.3.

The hyperparameters for the GNN baselines were taken from the Exphormer repository [21] for GINE and GatedGCN on the datasets PascalVOC-SP and Peptides-Func. For all other model/dataset combinations, we extrapolated the hyperparameters from these configurations, again adapting the hidden dimension to fit the same parameter budget as the Transformer models. The hidden dimension, as well as the

resulting number of parameters for each model is summarized in Table A.3. The other hyperparameters can be found in the configuration files in our repository.

Table A.2: The hyperparameters used for the graph Transformer models on the four small-graph datasets.

HP/Dataset	**Cifar10**	**MalNet-Tiny**	**PascalVOC-SP**	**Peptides-Func**
Metric	Accuracy	Accuracy	F1 score	AP
PE/SE type	ESLapPE	Local Degree Profile	LapPE	LapPE
Optimizer	adamW	adamW	adamW	adamW
Loss function	CE	CE	weighted CE	CE
# Epochs	100	150	300	200
Batch size	16	16	32	128
LR	1e-3	5e-4	5e-4	3e-4
LR schedule	cosine	cosine	cosine	cosine
# Warmup epochs	5	10	10	10
Weight decay	1e-5	1e-5	0	0
# GPS layers	5	5	4	8
MPNN layer	GatedGCN	GatedGCN	GatedGCN	GatedGCN
# Attention heads	4	4	4	4
Hidden dimension	variable	variable	variable	variable
Dropout	0.1	0	0.15	0.12
Attention dropout	0.1	0.2	0.5	0.5
# Input FCLs	0	1	0	0
# Output FCLs	2	3	3	1
Graph pooling	mean	max	–	mean
Expander Degree	5	5	5	5
# Virtual Nodes	1	4	0	1
Key-query dimension	16	16	16	16
Value dimension	9	16	22	17
k	10	10	10	10

Table A.3: The hidden dimension and number of parameters for each model/dataset combination on the four small-graph datasets. The first number in each cell is the hidden dimension, the second number is the number of parameters.

Model/Dataset	**Cifar10**	**MalNet-Tiny**	**PascalVOC-SP**	**Peptides-Func**
GCN	128 / 103178	160 / 287045	220 / 496945	296 / 497202
GINE	94 / 100778	122 / 286949	166 / 505875	214 / 500108
GAT	126 / 101314	160 / 289605	220 / 500465	296 / 500162
GatedGCN	61 / 101209	81 / 289418	108 / 502497	136 / 493162
GPS + BigBird	40 / 128335	60 / 289385	88 / 492941	64 / 514778
GPS + Performer	20 / 123575	40 / 286125	48 / 500733	44 / 513046
GPS + Transformer	40 / 111735	64 / 286725	96 / 510453	68 / 504110
Exphormer	40 / 111095	64 / 286277	96 / 509013	68 / 502206
k-MIP-GT	36 / 113375	64 / 296921	96 / 486433	68 / 512450

A.3 Hyperparameters: Efficiency

The efficiency experiment in section 6.2 is conducted in a controlled environment where we measure the efficiency of the attention mechanism in isolation. We sample keys, queries, and values from a normal distribution and measure the time it takes to compute the attention scores for a varying number of tokens. The hyperparameters used for this experiment are summarized in Table A.4.

Table A.4: The hyperparameters used for the efficiency experiment.

Hyperparameter	**Value**
Batch size	1
# **Heads**	1
# **Tokens**	variable
Key-query dimension	10
Value dimension	50
k	10

A.4 Hyperparameters: Ball Tree Search

The Ball Tree Search experiment in section 6.3 is conducted in a controlled environment where we study the properties and performance of the Ball Tree Search algorithm in isolation. We sample keys, queries, and values from a normal distribution and measure the time it takes to compute the attention scores for a varying number of tokens. In the curse of dimensionality experiment, we vary the number of tokens and study the number of nodes searched per query. In the efficiency comparison experiment, we compare the time it takes to compute the attention scores for a varying number of tokens.

The only difference between the set-ups of both experiments is the maximum leaf size in the construction of the ball tree. In the curse of dimensionality experiment, we set the maximum leaf size to 1, in order to get representative results for the amount of ball tree nodes searched per query. In the efficiency comparison experiment, we set the maximum leaf size to 100, which allows for higher levels of parallelism and is closer to the typical use case of the Ball Tree Search algorithm.

Table A.5: The hyperparameters used for the curse of dimensionality experiment.

Hyperparameter	**Value**
Batch size	1
# **Heads**	1
# **Tokens**	variable
Key-query dimension	variable
Value dimension	50
k	10
Max. leaf size	1

Table A.6: The hyperparameters used for the efficiency comparison experiment.

Hyperparameter	**Value**
# **Batch size**	1
# **Heads**	1
# **Tokens**	variable
Key-query dimension	variable
Value dimension	50
k	10
Max. leaf size	100

A.5 Hyperparameters: Influence of k

For the influence of k experiment, we used the same hyperparameters for k-MIP-GT as in the experiment from section 6.1. The only difference is that we vary the value of k among the set $\{1, 2, 5, 10, 20, 30, 50, 100\}$. The exact hyperparameters are summarized in Table A.7.

Table A.7: The hyperparameters used for all k-MIP-GTs in the influence of k experiment.

HP/Dataset	**Cifar10**	**MalNet-Tiny**	**PascalVOC-SP**	**Peptides-Func**
Metric	Accuracy	Accuracy	F1 score	AP
PE/SE type	ESLapPE	Local Degree Profile	LapPE	LapPE
Optimizer	adamW	adamW	adamW	adamW
Loss function	CE	CE	weighted CE	CE
# **Epochs**	100	150	300	200
Batch size	16	16	32	128
LR	1e-3	5e-4	5e-4	3e-4
LR schedule	cosine	cosine	cosine	cosine
# **Warmup epochs**	5	10	10	10
Weight decay	1e-5	1e-5	0	0
# **GPS layers**	5	5	4	8
MPNN layer	GatedGCN	GatedGCN	GatedGCN	GatedGCN
# **Attention heads**	4	4	4	4
Hidden dimension	36	64	88	68
Dropout	0.1	0	0.15	0.12
Attention dropout	0.1	0.2	0.5	0.5
# **Input FCLs**	0	1	0	0
# **Output FCLs**	2	3	3	1
Graph pooling	mean	max	–	mean
Key-query dimension	16	16	16	16
Value dimension	9	16	22	17
k	variable	variable	variable	variable
# **parameters**	113375	296921	486433	512450

A.6 Hyperparameters: Influence of d_{kq}

For the influence of d_{kq} experiment, we used the same hyperparameters for k-MIP-GT as in the experiment from section 6.1. The only difference is that we vary the value of d_{kq} among the set $\{1, 2, 5, 10, 20, 30, 50, 100\}$ and vary the hidden dimension of the model slightly to fit the same parameter budget. The exact hyperparameters are summarized in Table A.8.

Table A.8: The hyperparameters used for all k-MIP-GTs in the influence of d_{kq} experiment.

HP/Dataset	Cifar10	MalNet-Tiny	PascalVOC-SP	Peptides-Func
Metric	Accuracy	Accuracy	F1 score	Avg. Prec.
PE/SE type	ESLapPE	Local Degree Profile	LapPE	LapPE
Optimizer	adamW	adamW	adamW	adamW
Loss function	CE	CE	weighted CE	CE
# Epochs	100	150	300	200
Batch size	16	16	32	128
LR	1e-3	5e-4	5e-4	3e-4
LR schedule	cosine	cosine	cosine	cosine
# Warmup epochs	5	10	10	10
Weight decay	1e-5	1e-5	0	0
# GPS layers	5	5	4	8
MPNN layer	GatedGCN	GatedGCN	GatedGCN	GatedGCN
# Attention heads	4	4	4	4
Hidden dimension	36-44	64-72	96-108	68-76
Dropout	0.1	0	0.15	0.12
Attention dropout	0.1	0.2	0.5	0.5
# Input FCLs	0	1	0	0
# Output FCLs	2	3	3	1
Graph pooling	mean	max	–	mean
Expander Degree	5	5	5	5
# Virtual Nodes	1	4	0	1
Key-query dimension	variable	variable	variable	variable
Value dimension	9-11	16-18	24-27	17-19
k	10	10	10	10

A.7 Hyperparameters: Scaling to Datasets with Larger Graphs

Table A.9 displays the hyperparameters used in the Transformer-based models of the first experiment of section 6.1.

For Amazon-Computer these hyperparameters are almost exactly the configuration of the Exphormer model from the Exphormer paper [21], and this configuration was used for all Transformer-based models. The only modifications were modifying the hidden dimension of each model to fit a parameter budget of 300K parameters, and using GatedGCN as MPNN layer instead of GCN, to match the configurations used in the small-graph experiments.

For ShapeNet-Part and S3DIS, the hyperparameters are roughly based on those of the MalNet-Tiny dataset, as that was the dataset from section 6.1 with the largest

graphs. Importantly, we did not use a positional encoding, as the coordinates of each node in the point cloud are included in the node features, and they already encode the spatial information. We used the same hyperparameters for each graph Transformer model, except for the hidden dimension, which was adapted to fit a parameter budget of 300K parameters.

For ModelNet10, we had to make some adjustments to the hyperparameters in the other point cloud datasets, as the large graphs in the datasets led to memory issues. We reduced the number of layers to 4 for all models, and adapted the hidden dimension to fit a parameter budget of 100K parameters.

The hyperparameters for the GNN baselines are based on those for MalNet-Tiny as well. For each method, we created both a 5-layer version and a 10-layer version (except for ModelNet, where we only created a 5-layer version), and selected the one with the best validation performance. The hidden dimension was adapted to fit the same parameter budget as the Transformer models. All hidden dimensions and number of parameters for each model/dataset combination are summarized in Table A.10. All other hyperparameters can be found in the configuration files in our repository.

Table A.9: The hyperparameters used for the graph Transformer models on four datasets with increasingly larger graphs.

HP/Dataset	**Amazon-Computer**	**ShapeNet-Part**	**S3DIS**	**ModelNet10**
Metric	Accuracy	F1 score	mIoU	Accuracy
PE/SE type	RWSE	none	none	none
Optimizer	adamW	adamW	adamW	adamW
Loss function	CE	weighted CE	weighted CE	CE
# Epochs	150	120	120	120
Batch size	32	16	16	16
LR	1e-3	5e-4	5e-4	5e-4
LR schedule	cosine	cosine	cosine	cosine
# Warmup epochs	5	10	10	10
Weight decay	1e-3	1e-5	1e-5	1e-5
# GPS layers	4	8	8	4
MPNN layer	GatedGCN	GatedGCN	GatedGCN	GatedGCN
# Attention heads	2	4	4	4
Hidden dimension	variable	variable	variable	variable
Dropout	0.4	0.1	0.1	0.4
Attention dropout	0.8	0	0	0
# Input FCLs	1	0	0	0
# Output FCLs	1	2	2	2
Graph pooling	mean	mean	mean	mean
Expander Degree	3	3	3	3
# Virtual Nodes	0	3	3	3
Key-query dimension	16	5	5	5
Value dimension	18	14	14	12
k	10	10	10	10

Table A.10: The hidden dimension and number of parameters for each model/-dataset combination on four datasets with increasingly larger graphs. The first number in each cell is the hidden dimension, the second number is the number of parameters.

Model/Dataset	**Amazon-Computer**	**ShapeNet-Part**	**S3DIS**	**ModelNet10**
GCN	144 / 295442	220 / 305603	162 / 295987	122 / 91876
GINE	112 / 296562	163 / 303556	163 / 297977	94 / 99650
GAT	143 / 294350	220 / 307830	162 / 299227	122 / 93096
GatedGCN	76 / 295694	76 / 305950	76 / 303329	61 / 100477
GPS + BigBird	64 / 300290	48 / 291266	48 / 289597	40 / 102730
GPS + Performer	60 / 301550	28 / 292566	28 / 291577	20 / 98810
GPS + Transformer	68 / 298214	–	–	–
Exphormer	80 / 296610	52 / 295046	52 / 293241	44 / 107458
k-MIP-GT	72 / 286986	56 / 284122	56 / 282181	48 / 107306

A.8 Hyperparameters: An Approximation for Full Attention?

For the experiments in section 6.7, we worked with the k-MIP-GT that was trained on the MalNet-Tiny dataset in section 6.1. The hyperparameters used for the training of this model have already been summarized in Table A.2.

A.9 Hyperparameters: Inspecting the Attention Graphs

For the experiments in section 6.7, we worked with the k-MIP-GT that was trained on the Cifar10 dataset in section 6.1. The hyperparameters used for the training of this model have already been summarized in Table A.2.

B | Extra Visualizations

B.1 Extra Attention Graph Visualisations

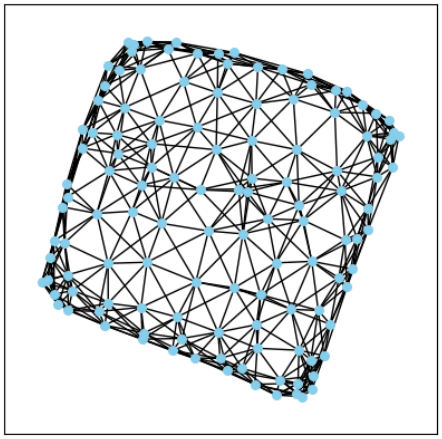


Figure B.1: The second input graph in the test split of the Cifar10 dataset. The directionality of the edges was omitted for clarity, as most edges are bidirectional.

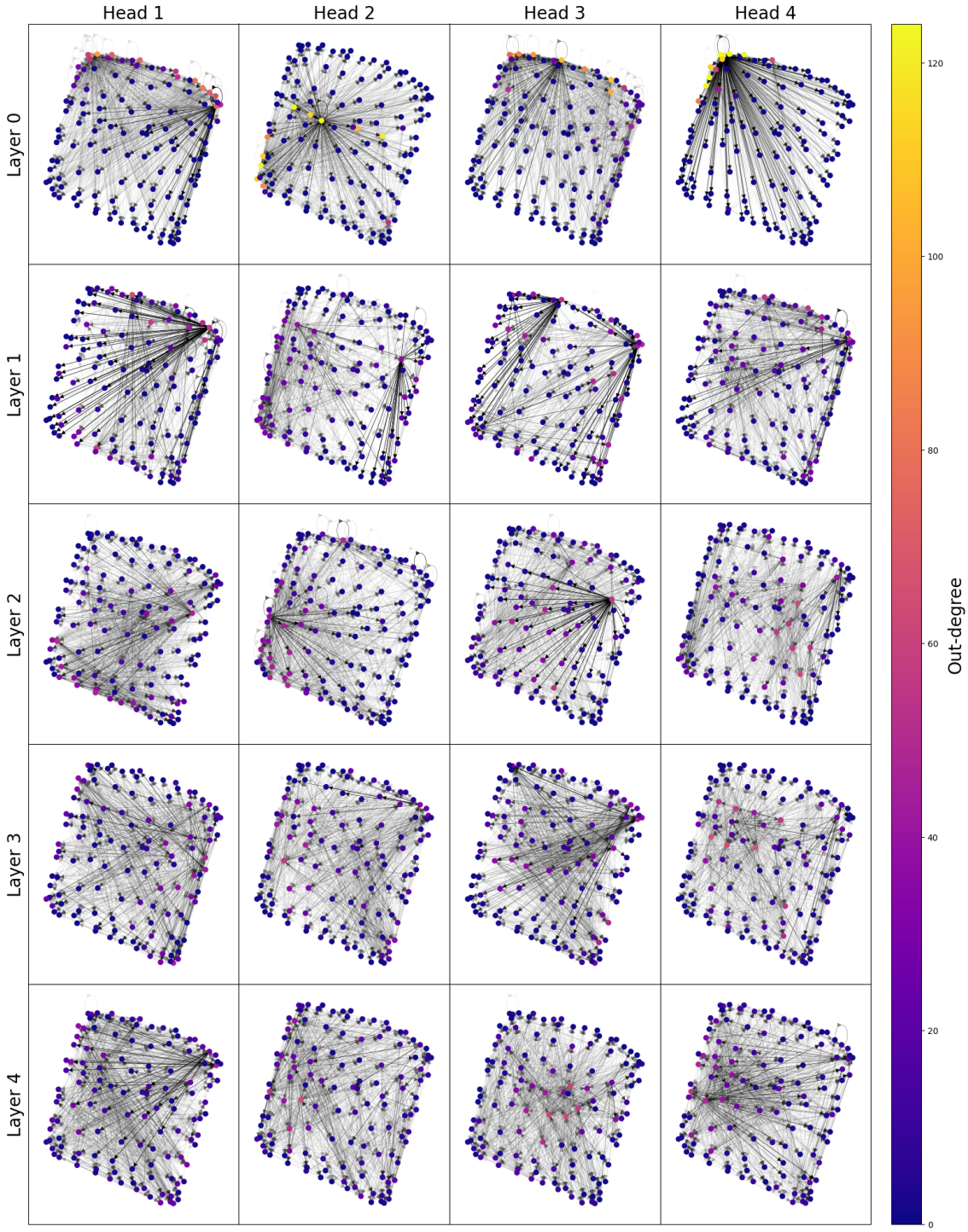


Figure B.2: The learnt attention graphs of each head and each layer in a trained k-MIP-GT, for the second input graph in the test split of the Cifar10 dataset. The edge alpha values are proportional to the corresponding attention weights. Each node is coloured according to its out-degree.

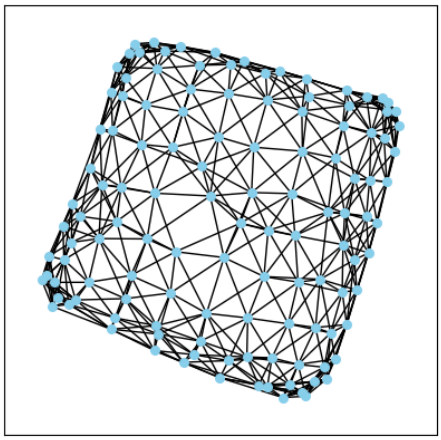


Figure B.3: The third input graph in the test split of the Cifar10 dataset. The directionality of the edges was omitted for clarity, as most edges are bidirectional.

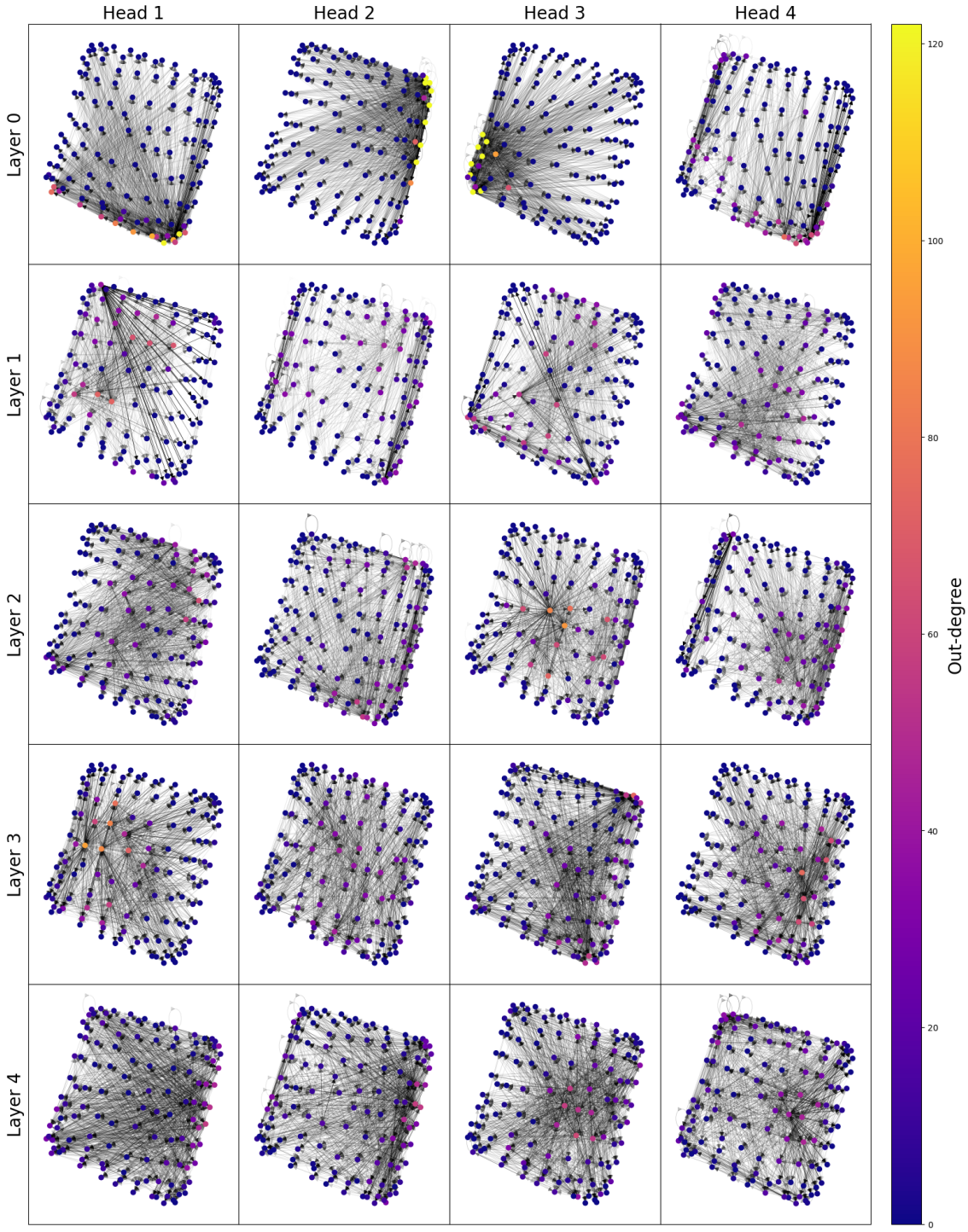


Figure B.4: The learnt attention graphs of each head and each layer in a trained k-MIP-GT, for the third input graph in the test split of the Cifar10 dataset. The edge alpha values are proportional to the corresponding attention weights. Each node is coloured according to its out-degree.

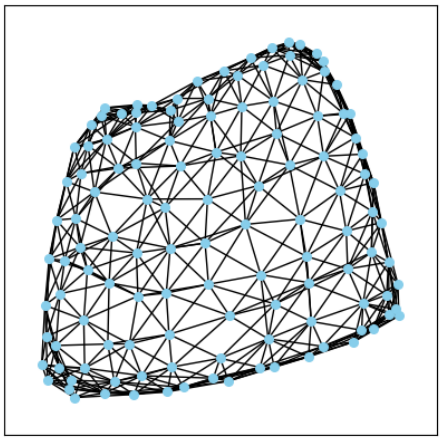


Figure B.5: The fourth input graph in the test split of the Cifar10 dataset. The directionality of the edges was omitted for clarity, as most edges are bidirectional.

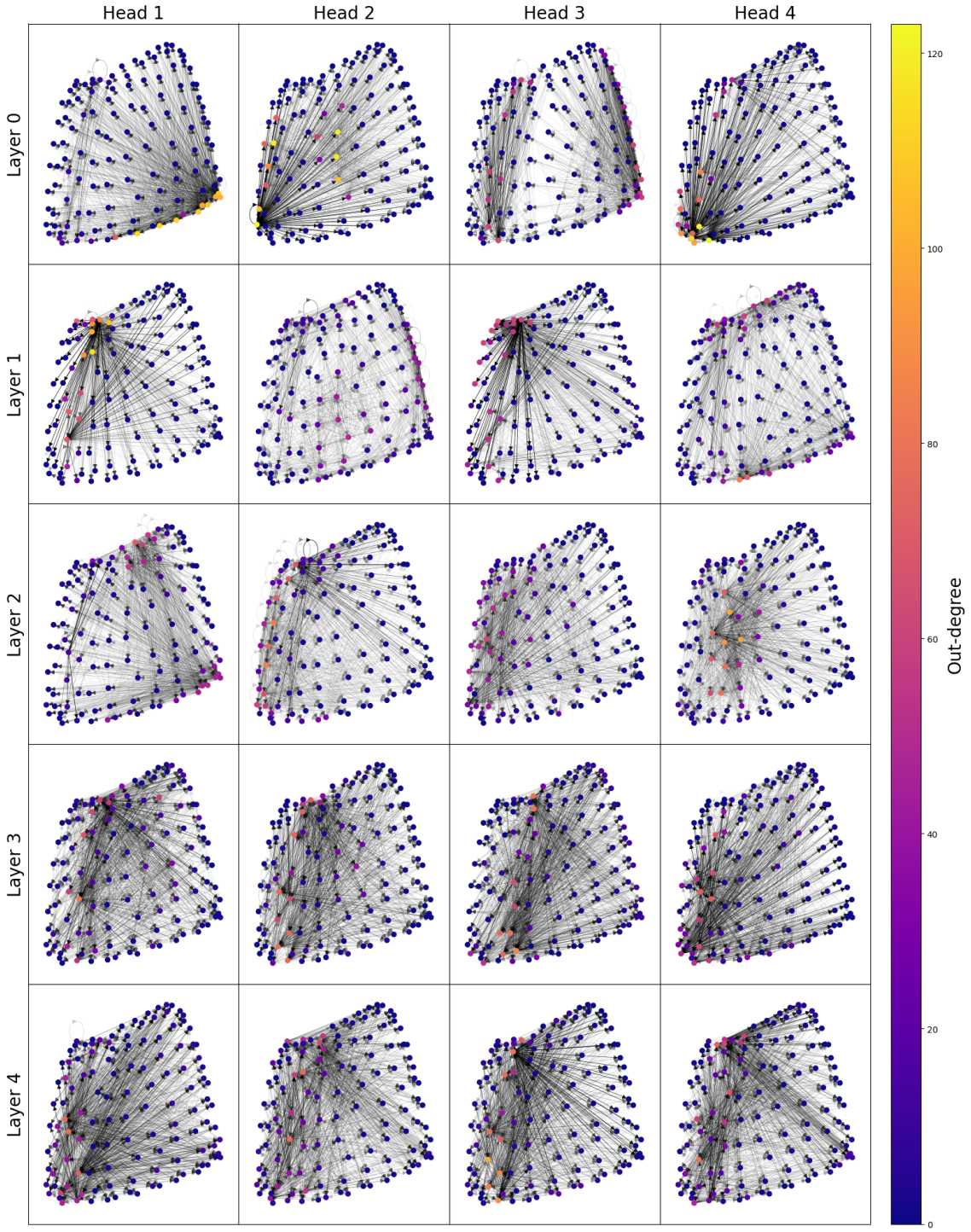


Figure B.6: The learnt attention graphs of each head and each layer in a trained k-MIP-GT, for the fourth input graph in the test split of the Cifar10 dataset. The edge alpha values are proportional to the corresponding attention weights. Each node is coloured according to its out-degree.