From block-Toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked Transformers

Krzysztof Choromanski [*1 2] **Han Lin** [*2] **Haoxian Chen** [*2] **Tianyi Zhang** [2] **Arijit Sehanobish** [3] **Valerii Likhosherstov** [4] **Jack Parker-Holder** [5] **Tamas Sarlos** [6] **Adrian Weller** [4 7] **Thomas Weingarten** [8]

Abstract

In this paper we provide, to the best of our knowledge, the first comprehensive approach for incorporating various masking mechanisms into Transformers architectures in a scalable way. We show that recent results on linear causal attention (Choromanski et al., 2021) and log-linear RPE-attention (Luo et al., 2021) are special cases of this general mechanism. However by casting the problem as a topological (graph-based) modulation of unmasked attention, we obtain several results unknown before, including efficient d-dimensional RPE-masking and graph-kernel masking. We leverage many mathematical techniques ranging from spectral analysis through dynamic programming and random walks to new algorithms for solving Markov processes on graphs. We provide a corresponding empirical evaluation.

1. Introduction & Related Work

Transformers (Vaswani et al., 2017; Brown et al., 2020; Devlin et al., 2019) have revolutionized machine learning by reintroducing an *attention mechanism* explicitly modeling complicated relationships between elementary ingredients of the ML models' inputs, e.g. words for text data, or patches/pixels for the image data (Han et al., 2020; Dosovitskiy et al., 2021). Crucially, attention quantifies these relationships via dynamic weights that depend on the input data. This architectural solution is the strength and at the same time the weakness of Transformer models. An attention matrix scales quadratically in the length of the input sequence, making corresponding computations prohibitively expensive for longer inputs.

Several solutions were proposed to address this limitation. Local attention (Vaswani et al., 2021; Parmar et al., 2019) explicitly narrows down the attention context to a fixed-size window, effectively zeroing out most attention weights. In applications where long-range attention is crucial (e.g. protein modeling), other techniques were introduced. These include: (1) pooling mechanisms compressing sequences to shorter-ones agglomerating multiple-tokens signal (Avsec et al., 2021; Dai et al., 2020), (2) hashing/clustering methods sparsifying attention by giving up attention modeling for tokens from different learnable hash-buckets/clusters (Kitaev et al., 2020; Roy et al., 2021), (3) low-rank/kernel-based methods decomposing the attention matrix (Choromanski et al., 2020; 2021; Katharopoulos et al., 2020; Peng et al., 2021; Xiong et al., 2021) and other (Qin et al., 2022).

Masking is a powerful mechanism altering the attention matrix by incorporating structural inductive bias. Flagship examples include (1) *causal attention*, applied in generative Transformers (Yang et al., 2019), where the arrow of time induces token-ordering with tokens not attending to their successors in the sequences, (2) *relative positional encoding* (RPE, Shaw et al., 2018) reducing interactions between distant tokens (but via a much more general mechanism than local attention) and (3) *graph attention* incorporating topological signal from the graph (Ying et al., 2021b; Velickovic et al., 2018). RPE-mechanisms were shown to significantly improve speech models (Pham et al., 2020; Zhou et al., 2019) and masks obtained from shortest-path length matrices were recently demonstrated to close the gap between the best customized graph neural networks models and Transformers (Ying et al., 2021a). Straightforward application of the masking mechanism requires materialization of the attention matrix and consequently - impractical quadratic time complexity for long input sequences (or large graphs).

In this paper we aim to answer the following question: *Under which conditions can masking be incorporated into attention mechanisms in a scalable way, i.e. in sub-quadratic time complexity in the number of input tokens?*

So far this question was answered only partially. Causality was incorporated in linear time into linear low-rank attention via the so-called *prefix sum mechanism* by Choroman-

[*]Equal contribution [1]Google Brain Robotics [2]Columbia University [3]Independent Researcher [4]University of Cambridge [5]University of Oxford [6]Google Research [7]The Alan Turing Institute [8]Google. Correspondence to: Krzysztof Choromanski <kchoro@google.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

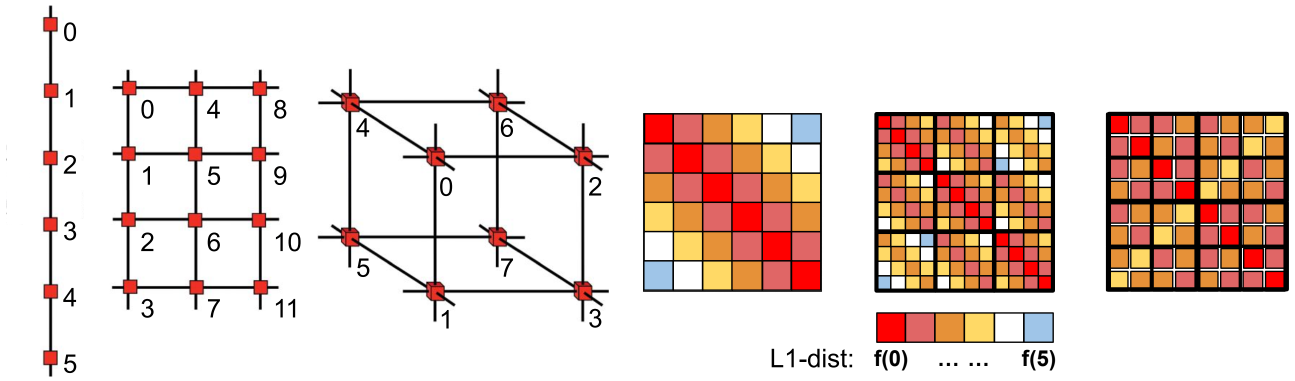

Figure 1. **RPEs & Beyond:** The (i, j)-entry of the regular RPE-mask is a (learnable) function f of the distance $i - j$ between the ith and jth token in the input sequence that can be interpreted as a 1d-grid, thus it has the so-called *Toeplitz* structure (first graph and colored-matrix in the figure). The proposed d-dimensional RPE acts on the d-dimensional grid input with the length of the shortest path $d(i, j)$ between node i and j in the gird replacing expression $i - j$ in the corresponding mask ($d = 2$ includes image input and $d = 3$, video input, see: second graph/matrix and third graph/matrix respectively). The corresponding mask is no longer Toeplitz, but is *d-level block-Toeplitz*. Interestingly, all these matrix classes support fast matrix-vector multiplication (via Fast Fourier Transform) and thus, based on our first result, corresponding masked low-rank attention can be performed in sub-quadratic time (see Sec. 3.2).

ski et al. (2021). The same was proven recently for the special class of *stochastic RPEs* (Liutkus et al., 2021). Even more recently, a log-linear algorithm (applying Fast Fourier Transform) for incorporating general RPEs into low-rank attention was given by Luo et al. (2021). All these results leverage low-rank attention since so far that was the only known scalable mechanism which can approximate in particular regular dense softmax attention. Hence, the starting point of our analysis is also a low-rank attention model.

Our contributions in this paper are as follows:

1. We answer the above question in Sec. 3 by providing a surprisingly simple characterization of the efficient masking mechanisms: *as long as the masking-matrix (element-wise multiplied with the regular attention matrix) supports sub-quadratic matrix-vector multiplication, the corresponding mechanism can be incorporated into low-rank attention in sub-quadratic time*. Interestingly, as we explain later, this result includes all mentioned partial results as special cases.
2. We present multiple consequences, leading in particular to novel scalable d-dimensional RPE mechanisms that can be applied in image and video processing (see Fig. 1 and Sec. 3.2), efficient implementations for the low-rank attention with *padding* and *packing* mechanisms (in common practical use for regular Transformers) (Sec. 3), and new scalable graph-based attention masking applying shortest-path signal and graph-diffusion kernels (Sec. 3.3, Sec. 4).
3. Using our developed theory, we introduce a new masked-attention ML-model called *graph kernel attention Transformer* (GKAT, Sec. 4.2), and conduct comprehensive comparisons against **nine** other SOTA graph neural networks (GNNs) in Sec. 5.

We cast the masking problem as a topological (graph-based) modulation of unmasked attention, and leverage many mathematical techniques ranging from spectral analysis through dynamic programming on trees and random walks to new algorithms for solving Markov processes on graphs. The proofs of all the theoretical results are given in the Appendix.

2. Preliminaries

We introduce notation used throughout the paper.

Denote by L the number of input tokens. The attention used in a regular Transformer linearly projects their representations into three learnable matrices $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{L \times d_{QK}}$, $\mathbf{V} \in \mathbb{R}^{L \times d}$ called *queries*, *keys* and *values* respectively.

Definition 2.1 (general masked attention)**.** *General masked softmax attention* is of the following form, where $\mathbf{N} \in \mathbb{R}^{L \times L}$ is the *logits-mask*, and $\mathbf{A} \in \mathbb{R}^{L \times L}$ is the so-called *masked attention matrix* (MAM):

$$\begin{aligned} \mathrm{Att}_{\mathrm{SM}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{N}) = \mathbf{D}^{-1}\mathbf{A}\mathbf{V}, \\ \mathbf{A} = \exp(\mathbf{N} + \mathbf{Q}\mathbf{K}^{\top}/\sqrt{d_{QK}}), \quad \mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1}_L). \end{aligned} \tag{1}$$

Here $\exp(\cdot)$ is applied element-wise, $\mathbf{1}_L$ is the all-ones vector of length L, and $\mathrm{diag}(\cdot)$ is a diagonal matrix with the input vector as the diagonal. The time complexity of computing (1) is $O(L^2 d)$. The above is a special instantiation of the *general masked kernel attention* which is defined as:

$$\begin{aligned} \mathrm{Att}_{\mathrm{K}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}) = \mathbf{D}^{-1}\mathbf{A}\mathbf{V}, \\ \mathbf{A} = \mathbf{M} \odot \mathcal{K}(\mathbf{Q}, \mathbf{K}), \quad \mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1}_L), \end{aligned} \tag{2}$$

where $\odot$ denotes the element-wise (Hadamard) matrix product, $\mathrm{K} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is some kernel function and $\mathcal{K}(\mathbf{Q}, \mathbf{K})$ is a kernel matrix defined as: $\mathcal{K}(\mathbf{Q}, \mathbf{K})_{i,j} = \mathrm{K}(\mathbf{q}_i^{\top}, \mathbf{k}_j^{\top})$ for the ith row $\mathbf{q}_i$ of $\mathbf{Q}$ and the jth row $\mathbf{k}_j$ of $\mathbf{K}$ respectively. We call $\mathbf{A}' = \mathcal{K}(\mathbf{Q}, \mathbf{K})$ the unmasked attention matrix (UAM). The softmax attention can be obtained from the kernel one by taking: $\mathrm{K}(\mathbf{x}, \mathbf{y}) \overset{\mathrm{def}}{=} \exp(\frac{\mathbf{x}^\top \mathbf{y}}{\sqrt{d_{QK}}})$ (the so-called *softmax kernel*) and $\mathbf{M} \overset{\mathrm{def}}{=} \exp(\mathbf{N})$ (element-wise exponentiation).

Low-rank attention methods provide (approximate) attention computation in time linear in the length L of the input sequence if no masking is applied (i.e. $\mathbf{M}$ is all-ones) and kernel K admits (at least in expectation) a dot-product decomposition, i.e. $\mathrm{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})^\top \phi(\mathbf{y})]$ for some (usually randomized) mapping: $\phi : \mathbb{R}^{d_{QK}} \rightarrow \mathbb{R}^m$ (and some $m > 0$). Such a decomposition (in fact more than one!) exists in particular for the softmax kernel used in most applications of regular Transformers. We call $\phi(\mathbf{u})$ a *(random) feature map* (RFM) for $\mathbf{u} \in \mathbb{R}^d$. For $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{L \times m}$ with rows given as $\phi(\mathbf{q}_i^\top)^\top$ and $\phi(\mathbf{k}_i^\top)^\top$ respectively, RFM-based kernel linearization leads directly to the efficient unmasked attention mechanism of the form:

$$\begin{aligned} \widehat{\mathrm{Att}_{\mathrm{K}}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \widehat{\mathbf{D}}^{-1}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{V})), \\ \widehat{\mathbf{D}} &= \mathrm{diag}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{1}_L)). \end{aligned} \tag{3}$$

Here $\widehat{\mathrm{Att}_{\mathrm{K}}}$ stands for the approximate attention and brackets indicate the order of computations. It is easy to see that such a mechanism is characterized by time complexity $O(Lmd)$ as opposed to $O(L^2 d)$ for regular attention. If $m \ll L$, computational gains are obtained.

3. Fast matrix-vector product is all you need

Our first result, a natural extension of the theoretical analysis by Luo et al. (2021), shows that as long as mask $\mathbf{M} \in \mathbb{R}^{L \times L}$ supports sub-quadratic matrix-vector multiplication, it can be incorporated into low-rank attention in sub-quadratic time. This is explained in Lemma 3.1.

Algorithm 1 General Efficient Low-Rank Masked Attention

```
Input: Query/key matrices: $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{L \times d_{QK}}$, value matrix $\mathbf{V} \in \mathbb{R}^{L \times d}$, mask $\mathbf{M} \in \mathbb{R}^{L \times L}$, procedure $\mathrm{FastMult}_{\mathbf{M}} : \mathbb{R}^L \rightarrow \mathbb{R}^L$ calculating $\mathbf{Mx}$ (or its approximation) for the input $\mathbf{x} \in \mathbb{R}^L$, kernel feature map: $\phi : \mathbb{R}^{d_{QK}} \rightarrow \mathbb{R}^m$. $\mathrm{vec}(\cdot)$ denotes vectorization.
Output: Masked low-rank attention embeddings using $\phi$.
1. Compute matrices $\mathbf{V}^1 \in \mathbb{R}^{L \times (md)}$, $\mathbf{V}^2 \in \mathbb{R}^{L \times m}$ with rows defined as: $\mathbf{V}^1_{i:} = \mathrm{vec}(\phi(\mathbf{k}_i^\top)\mathbf{v}_i)$, $\mathbf{V}^2_{i:} = \phi(\mathbf{k}_i^\top)^\top$, where $\mathbf{k}_i/\mathbf{v}_i$ stands for the ith row of $\mathbf{K}/\mathbf{V}$.
2. Take $\tilde{\mathbf{D}}^1 = [\mathrm{FastMult}_{\mathbf{M}}(\mathbf{V}^1_{:1}), ..., \mathrm{FastMult}_{\mathbf{M}}(\mathbf{V}^1_{:md})] \in \mathbb{R}^{L \times md}$, $\tilde{\mathbf{D}}^2 = [\mathrm{FastMult}_{\mathbf{M}}(\mathbf{V}^2_{:1}), ..., \mathrm{FastMult}_{\mathbf{M}}(\mathbf{V}^2_{:m})] \in \mathbb{R}^{L \times m}$ for $\mathbf{V}^{1/2}_{:i}$ denoting ith column of $\mathbf{V}^{1/2}$.
3. Output the embedding $\mathbf{r}_i$ of the ith tokens as: $\mathbf{r}_i = \frac{\phi(\mathbf{q}_i^\top)^\top \mathrm{devec}(\tilde{\mathbf{D}}^1_{i:})}{\phi(\mathbf{q}_i^\top)^\top (\tilde{\mathbf{D}}^2_{i:})^\top}$, where $\mathbf{q}_i$ is the ith row of $\mathbf{Q}$ and $\mathrm{devec}(\cdot)$ devectorizes its input back to $\mathbb{R}^{m \times d}$.
```

Lemma 3.1 (Tractable Mask Lemma). *Assume that mask $\mathbf{M} \in \mathbb{R}^{L \times L}$ from Definition 2.1 supports matrix-vector multiplication in time $T_{\mathbf{M}}(L)$. Then the general masked kernel attention algorithm with mask $\mathbf{M}$ can be implemented in time $O((T_{\mathbf{M}}(L) + L)md)$.*

The algorithm is given in the algorithmic box 1. We analyze it in the Appendix, but the intuition is that, in the unmasked low-rank setting, attention embeddings could be obtained from the action of $\phi(\mathbf{q}_i)$ on the fixed (token-independent) matrix of shape $\mathbb{R}^{m \times d}$ summarizing all the tokens, whereas in the masked case the matrix depends on each token, but can be obtained from mask-vector products.

Causal attention: Note that the prefix-sum algorithm from (Choromanski et al., 2021) is a special instantiation of Algorithm 1. Indeed, causality is encoded by the lower-triangular mask $\mathbf{M}$ such that: $\mathbf{M}_{i,j} = 1$ for $j \leq i$ and $\mathbf{M}_{i,j} = 0$ otherwise. Every product $\mathbf{Mx}$ is trivially a vector of prefix sums: $\mathbf{x}_1 + ... + \mathbf{x}_i$ for $i = 1, ..., L$ and thus can be computed in time $O(L)$.

Packing & Padding: Both masking mechanisms are standard Transformers' techniques used to optimize attention computation on TPUs. The former packs multiple sequences in one *super-sequence*. Mask is used here to prevent cross-sequence attention. The latter adds *fake* tokens at the end of the legitimate input sequence (used if input's length varies). Mask is used here to prevent attention to fake tokens. Both masks $\mathbf{M}$ trivially support linear matrix-vector multiplication (see: Fig. 2) and thus both packing and padding can be incorporated into low-rank attention in time linear in L.

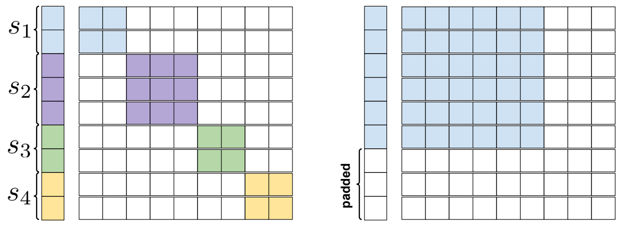

Figure 2. **Left**: Padding with the super-sequence consisting of four sequences and its corresponding mask $\mathbf{M}$. **Right:** Packing with three fake padded tokens and its corresponding mask $\mathbf{M}$. For both masks, colored entries are equal to one and non-colored are equal to zero. Both masks trivially support linear matrix-vector multiplication.

3.1. Mask M as graph topology encoder

From now on we will think about mask $\mathbf{M} \in \mathbb{R}^{L \times L}$ as a weighted adjacency matrix $\mathrm{Adj}(G)$ of some weighted graph $G = (V, E, W)$ with nodes/vertex-set V of size L, edge set E and edge-weight function $W : E \rightarrow \mathbb{R}$. Lemma 3.1 combined with this observation leads to several far-reaching conclusions regarding efficient incorporation of various masking mechanisms to Transformers, to which we devote the remaining part of our theoretical analysis.

3.2. D-dimensional Relative Positional Encodings

We need the following definition:

Definition 3.2 (block-Toeplitz matrices)**.** We say that a matrix $\mathbf{M} \in \mathbb{R}^{L \times L}$ is Toeplitz (or 1-level block Toeplitz) if there exists some $\xi : \mathbb{Z} \to \mathbb{R}$ such that $\mathbf{M}_{i,j} = \xi(i-j)$. We say that $\mathbf{M} \in \mathbb{R}^{L \times L}$ is d-level block-Toeplitz for $d \geq 2$ if $\mathbf{M} = (\mathbf{B}^{i,j})$ consists of block-matrices $\mathbf{B}^{i,j}$ taken from some set $\{\mathbf{A}_1, ..., \mathbf{A}_r\}$ of $(d-1)$-level block-Toeplitz matrices and if each block $\mathbf{B}^{i,j}$ is replaced with the index k of its corresponding matrix $\mathbf{A}_k$, a Toeplitz matrix is obtained.

Consider the unweighted (i.e. all-one edge-weights) 1d-grid graph G_{base} (see: left graph in Fig. 1) and a complete graph (i.e. with all possible edges) G obtained from it by defining each weight as $W_{i,j} = f(\text{dist}_{G_{\text{base}}}(i,j))$ for some (learnable) function $f : \mathbb{N} \to \mathbb{R}$ and where $\text{dist}_{G_{\text{base}}}(i,j)$ is the length of the shortest path between i and j in G_{base}. If we define $\mathbf{M} = \text{Adj}(G)$ then we get the regular RPE mechanism with the 1d-graph interpreted as the input sequence.

Note that $\mathbf{M}$ defined in such a way is Toeplitz and thus supports $O(L\log(L))$ matrix-vector multiplication via Fast Fourier Transform (FFT). Thus low-rank RPE-masked attention can be conducted in $O(Ldm\log(L))$ time. This was the observation of Luo et al. (2021). What if we replace the 1d-grid with the d-dimensional grid and define $\mathbf{M}$ in the analogous way? The idea is to maintain the initial structure of the topologically more complicated input, e.g. 2d-grid for images (with nodes as patches or even individual pixels) or 3d-grid for videos (with 2d-slices as different frames).

There is a particularly elegant answer to this question:

Lemma 3.3 (d-dimensional RPEs)**.** *Consider the generalized RPE-mechanism for the d-dimensional grid input defined above. Then there exists an ordering of input nodes such that* $\mathbf{M}$ *is a d-level block-Toeplitz matrix (see: Fig. 1).*

Since d-level block-Toeplitz matrices support $O(L\log(L))$ matrix-vector multiplication via FFT for any fixed constant d (see Lee, 1986), Lemma 3.3 immediately leads to efficient corresponding masked attention computation.

3.3. More general graph-masks using shortest-paths

So far G_{base} was assumed to have a grid structure. What if we replace it with an arbitrary weighted graph? The following natural question arises: *Which condition does* G_{base} *and mapping* $f : \mathbb{R} \to \mathbb{R}$ *need to satisfy for the mask* $\mathbf{M} \overset{\text{def}}{=} [f(\text{dist}_{G_{\text{base}}}(i,j))]_{i,j=1,...,L}$ *to support sub-quadratic matrix-vector multiplication ?*

We call such a pair (G_{base}, f) *tractable*. From what we have said so far, we conclude that:

Corollary 3.4. *If* G_{base} *is an unweighted grid (of any dimensionality) then* (G_{base}, f) *is tractable for any* $f : \mathbb{R} \to \mathbb{R}$.

In several bioinformatics applications, e.g. molecular assembly trees (Artemova et al., 2011), the underlying input's topology is a forest (e.g. a tree). We prove the following:

Lemma 3.5. *If* G_{base} *is a forest and* $f(z) = \exp(\tau(z))$ *for affine mapping* τ*, then* (G_{base}, f) *is tractable and the related mask supports linear matrix-vector multiplication.*

Sketch of the proof: The efficient algorithm for computing $\mathbf{w} = \mathbf{M}\mathbf{x}$ in this case is an application of the dynamic programming method for rooted trees. The algorithm first computes for each node i the following expression: $s_i = \sum_{j \in \mathcal{T}_i} \exp(\tau(\text{dist}(i,j)))\mathbf{x}_j$, where $\mathcal{T}_i$ stands for the subtree rooted in i (in the bottom-up fashion from leaves to the fixed root). This is followed by the computation of the following expression: $\mathbf{w}_i = \sum_{j \in \mathcal{T}} \exp(\tau(\text{dist}(i,j)))\mathbf{x}_j$ for every node in the order from the root to the leaves (leveraging already computed s_i). Details are given in the Appendix and computations are illustrated in Fig. 3.

We find a comprehensive description of tractable (G_{base}, f) an exciting analytic and combinatorial open problem.

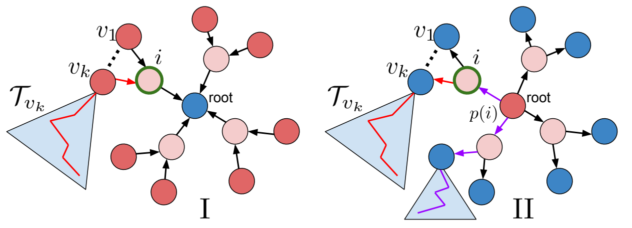

Figure 3. Illustration of sketch of the proof of Lemma 3.5. The directions of arrows show computation-flow. In phase I, s_i-terms are calculated in bottom-up fashion (from leaves to the root). The value of s_i involving paths in i-rooted subtrees (red path with discarded directions) is updated based on s_{v_k}-terms involving paths in subtrees $\mathcal{T}_{v_k}$. To complete calculations, in phase II paths to nodes outside of the i-rooted tree are considered (purple path with directions discarded). Their contribution is calculated from the already computed $\mathbf{w}_{p(i)}$ for the parent $p(i)$ of node i and s_i.

3.4. Low-rank masking

Note that in all previously considered cases, mask $\mathbf{M}$ is in general full-rank. However in several applications $\mathbf{M}$ can be assumed to have (at least in expectation) a low-rank decomposition, i.e.: $\mathbf{M} = \mathbb{E}[\mathbf{M}_1\mathbf{M}_2]$ for some (random) $\mathbf{M}_1 \in \mathbb{R}^{L \times r}$, $\mathbf{M}_2 \in \mathbb{R}^{r \times L}$ and some $0 < r \ll L$. A flagship example is the *stochastic RPE* mechanism presented in (Liutkus et al., 2021) corresponding to (logits-added) dot-product mask $\mathbf{N}$ translating to the softmax-kernel values mask $\mathbf{M}$. The latter one can be low-rank decomposed using any random feature map based softmax-kernel linearization mechanism, e.g. from (Choromanski et al., 2021). In such a case, matrix-vector product $\mathbf{v} = \mathbf{M}\mathbf{x}$ can be computed (approximately) as: $\tilde{\mathbf{v}} = (\mathbf{M}_1(\mathbf{M}_2\mathbf{x}))$ in time $O(Lr)$, leading to overall time complexity $O(Lmrd)$ of the attention module using mask $\mathbf{M}$.

4. Masking with graph kernels

Masks defined by shortest paths were shown to provide effective inductive bias for graph data (see Ying et al., 2021b), yet they cannot be interpreted as applying any valid kernel function on graph nodes and are very sensitive to small changes in the graph. A prominent class of kernel-functions $\mathrm{K} : V \times V \rightarrow \mathbb{R}$ defined on pairs of graphs nodes is the family of *graph-diffusion* or *heat* kernels (GDKs). Thus it is natural to identify masks $\mathbf{M}$ for input graph data G with the graph diffusion kernel matrices $\mathcal{K}_{\mathrm{K}} = [\mathrm{K}(i,j)]_{i,j=1,...,L}$. GDK is defined, for a hyperparameter $\lambda > 0$ and $\mathbf{X}^i$ denoting the ith power of matrix $\mathbf{X}$, as:

$$\mathcal{K}_{\mathrm{K}} = \exp(-\lambda \mathbf{T}) \overset{\mathrm{def}}{=} \sum_{i=0}^{\infty} \frac{(-\lambda)^i \mathbf{T}^i}{i!}, \tag{4}$$

where either: $\mathbf{T} = \mathbf{L}$ for the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathrm{Adj}(G)$ and $\mathbf{D} = \mathrm{diag}([\deg(i)]_{i=1}^{L})$; or $\mathbf{T} = \mathbf{L}\mathbf{D}^{-1}$ (normalized Laplacian case) or $\mathbf{T} = -\mathrm{Adj}(G)$.

GDK is related to the diffusion process (Kondor & Lafferty, 2002) which describes in particular heat propagation. In a vacuum, the solution of the partial differential heat equation is the Gaussian-kernel, and in graphs it leads to GDK. Nodes better connected with each other (graph diffusion kernel quantifies it via the number of different-length walks with longer walks exponentially-deprioritized) give rise to larger kernel values. Finally, $t = \frac{1}{\lambda}$ can be interpreted as time when the solution is taken. GDK imprints topological signal of the propagation medium via left heat-signature. As $t \rightarrow \infty$ the kernel "flattens" and the topological signal is lost.

Direct computation of the GDK matrix $\mathcal{K}_{\mathrm{K}}$ is of $O(L^3)$ time complexity, thus prohibitively expensive even for sparse input graphs. However, a key observation is that Lemma 3.1 teaches us that for efficient masked low-rank attention we only need to compute efficiently the action $\exp(-\lambda\mathbf{T})\mathbf{x}$ of $\mathcal{K}_{\mathrm{K}}$ on a given $\mathbf{x} \in \mathbb{R}^L$. This leads to our next result.

Theorem 4.1 (scalable Laplacian-GDK masking)**.** *Let a mask $\mathbf{M}$ be defined as $\mathbf{M} = \exp(-\lambda\mathbf{A})$, for $\mathbf{A} = \mathbf{L}$ or $\mathbf{A} = \mathbf{L}\mathbf{D}^{-1}$, where $\mathbf{L}$ is the Laplacian of the input graph, and $\mathbf{D} = \mathrm{diag}([\deg(i)]_{i=1}^{L})$. Then low-rank masked attention can be computed in time $\tilde{O}((|E| + L)\log(2 + \|\mathbf{A}\|_{\mathrm{F}})md)$, where $\tilde{O}$ hides $\mathrm{polylog}(L)$ factors, $|E|$ is the number of graph edges and $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm.*

The theorem is a consequence of Lemma 3.1 and Theorem 1.2 from (Orecchia et al., 2012). We see that if $|E| = o(L^2)$, the masked attention mechanism is sub-quadratic in L.

Low-rank attention & Markov processes with random initial distributions: As noted by Orecchia et al. (2012), the heat kernel matrix for $\mathbf{T} = \mathbf{L}\mathbf{D}^{-1}$ can be interpreted as the probability transition matrix of the discrete-time random walk where first the number of steps i is sampled from a Poisson distribution with mean λ, and then i steps of the natural random walk are performed on G. Looking at Algorithm 1, we conclude that here the low-rank structure of attention enables us to incorporate the GDK mask by solving that process in md initial (randomized) distributions over nodes (randomization coming from mapping ϕ) rather than in all L one-hot initial distributions (that would correspond to the reconstruction of the entire transition matrix).

Remark: The literature on efficiently computing the actions of matrix-exponentials (which is our main focus in this section) is very rich (Al-Mohy & Higham, 2011), partially because of straightforward applications in the theory of differential equations (Li et al., 2021). In principle, each of these methods can be used by our algorithm.

4.1. Grid with graph-diffusion kernels

If the underlying graph is a d-dimensional grid, then GDK with $\mathbf{T} = \mathbf{L}$ has a closed-form formula. The following is true (Kondor & Lafferty, 2002): $\mathrm{K}_{\mathrm{GDK}}(i,j) \propto (\tanh\lambda)^{\mathrm{dist}(i,j)}$ for the hyperbolic tangent $\tanh$. Thus, as in Sec. 3.2, the corresponding mask $\mathbf{M}$ is d-level block-Toeplitz and grid-induced GDK-masking can be incorporated into low-rank attention in $O(Lmd\log(L))$ time.

4.2. Low-rank masking strikes back for GDKs

We now propose a proxy of the GDK with $\mathbf{T} = \mathrm{Adj}(G)$ such that the corresponding kernel matrix admits (in expectation) low-rank decomposition as in Sec. 3.4. Thus, based on the theory we developed, the corresponding mask $\mathbf{M}$ can be efficiently incorporated into low-rank attention with no need to call efficient solvers for the actions of matrix exponentials. We call our graph kernel the *Random Walks Graph-Nodes Kernel* or RWGNK.

Intuitively, the value of the RWGNK for two nodes is given as a dot-product of two *frequency vectors* that record visits in graph nodes of random walks beginning in the two nodes of interest. More formally, for the hyperparameters $\lambda, \alpha \geq 0$, and two random walks $\omega(k)$, $\omega(l)$ with stopping probability $0 \leq p \leq 1$ (or of a fixed length) starting at k and l respectively, the RWGNK is given as:

$$\mathrm{K}_p^{\lambda,\alpha}(k,l) = \frac{\mathbb{E}_{\omega(k)}[f_k^{\omega(k),\lambda}]}{\|\mathbb{E}_{\omega(k)}[f_k^{\omega(k),\lambda}]\|_2^{\alpha}} \left(\frac{\mathbb{E}_{\omega(l)}[f_l^{\omega(l),\lambda}]}{\|\mathbb{E}_{\omega(l)}[f_l^{\omega(l),\lambda}]\|_2^{\alpha}}\right)^{\top}. \tag{5}$$

The (row) frequency vector $f_h^{\omega(h),\lambda}$ for $h \in \mathrm{V}$ is given as $f_h^{\omega(h),\lambda}(i) \overset{\mathrm{def}}{=} \sum_{e \in \mathrm{L}^{\omega(h)}(i)} \lambda^e$, where $\mathrm{L}^{\omega(h)}(i)$ is the set of lengths of those *prefix sub-walks* of a given random walk $\omega(h)$ that end at i (where the prefix sub-walk of the walk $(j_1, .j_2, ..., j_t)$ is any walk of the form $(j_1, ..., j_r)$ for some $r \leq t$ or an empty walk). Note that Eq. 5 leads to the desired representation of $\mathrm{K}_p^{\lambda,\alpha}$ as $\mathrm{K}_p^{\lambda,\alpha}(k,l) = \Psi(k)\Psi(l)^{\top}$,

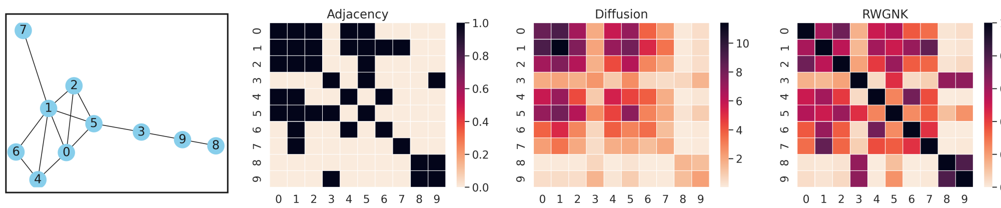

Figure 4. From left to right: unweighted graph G, its adjacency matrix $\mathrm{Adj}(\mathrm{G})$, its GDK matrix $\exp(\mathrm{Adj}(\mathrm{G}))$ and RWGNK-matrix with walk length of 3 and $\alpha = 1$. Colored cells measure the relationship among pairs of nodes (darker is stronger). The last two matrices can be thought of as continuous smoothings of $\mathrm{Adj}(\mathrm{G})$.

where $\Psi(h)$ is the renormalized expected frequency vector. In practice, expectations are replaced by Monte Carlo samplings over a few random walks, and vectors $\Psi(h)$ are not stored explicitly but in the form of weighted lookup tables.

Figure 4 compares RWGNK-induced mask with the regular GDK-mask and the adjacency matrix mask. We call a Transformer applying low-rank masked attention via RWGNK, a *Graph Kernel Attention Transformer* (or GKAT).

Next we explore the connection of RWGNKs with GDKs. We denote by $d_{\max}, d_{\min}$ the maximum and minimum degree of a vertex in G respectively.

Theorem 4.2 (RWGNKs count discounted numbers of walks)**.** *The following is true for the kernel matrix* $\mathcal{K}_p^{\lambda,\alpha}(\mathrm{G}) = [\mathrm{K}_p^{\lambda,\alpha}(k,l)]_{k,l\in\mathrm{V}(\mathrm{G})}$ *of the* RWGNK *kernel with* $0 \leq \lambda \leq 1$, $\alpha = 0$ *and* $0 < p < 1$ *for a graph* G *with vertex set* $\mathrm{V}(\mathrm{G})$ *of size* N *(element-wise matrix inequality):*

$$\Gamma\left(\frac{\rho}{d_{\max}}\mathrm{Adj}(\mathrm{G})\right) \leq \mathcal{K}_p^{\lambda,\alpha}(\mathrm{G}) \leq \Gamma\left(\frac{\rho}{d_{\min}}\mathrm{Adj}(\mathrm{G})\right), \tag{6}$$

where $\rho = (1-p)\lambda$ and $\Gamma(\mathbf{A}) = \sum_{i=0}^{\infty}(i+1)\mathbf{A}^i$. Using the fact that $\mathrm{Adj}^i(\mathrm{G})$ encodes the number of walks of length i between pairs of vertices in G, we conclude that $\mathrm{K}_p^{\lambda,0}(k,l) = \sum_{i=0}^{\infty} c_{k,l}^i r_{k,l}(i)$, where: $r_{k,l}(i)$ is the number of walks of length i between nodes: k and l and $\frac{\sqrt[i]{i+1}(1-p)\lambda}{d_{\max}} \leq c_{k,l} \leq \frac{\sqrt[i]{i+1}(1-p)\lambda}{d_{\min}}$. Note that values of GDK with parameter λ satisfy: $\mathrm{GDK}^{\lambda}(k,l) = \sum_{i=0}^{\infty}\tilde{c}^i(k,l)r_{k,l}(i)$, where: $\tilde{c}(k,l) = \frac{\lambda}{\sqrt[i]{i!}}$. In practice, it suffices to have random walks of fixed length (instead of taking $p > 0$) (see: Sec 5). Furthermore, by taking $\alpha > 0$ (e.g. $\alpha = 1$) we can guarantee that kernel values are bounded.

5. Experiments

We focus on the GKAT architecture introduced in Sec. 4.2 as a prominent instantiation of the general mechanism presented in this paper and experiments with 2-level block-Toeplitz masking mechanisms introduced in Sec. 3.2 for vision Transformers.

Regarding GKAT, we conducted exhaustive evaluations on tasks ranging from purely combinatorial to bioinformatics, and benchmarked **10** different methods. All these experiments were run on a single Tesla P100 GPU with 16GB memory. Experiments with vision Transformers were conducted on the ImageNet dataset.

5.1. Combinatorial Classification

In this section we focus on the problem of detecting local patterns in graphs. A model takes a graph G as an input and decides whether it contains some graph from the given family of graphs $\mathcal{H}$ as a subgraph (not necessarily induced) or is $\mathcal{H}$-free. This benchmark tests the abilities of different methods to solve purely combinatorial tasks.

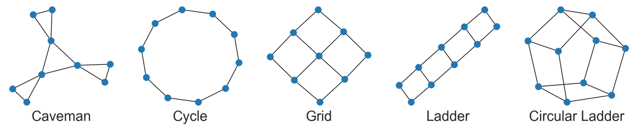

Figure 5. Five motifs (patterns) used in the first class of combinatorial classification experiments. For each pattern H, an algorithm is trained to distinguish between graphs G containing H and those that are H-free. A naive brute-force algorithm for conducting this has time complexity $\Omega(N^h)$, where h is the number of nodes of the motif, prohibitively expensive for all these motifs (since $h \geq 9$).

5.1.1. Erdős-Rényi Random Graph with Motifs

Data Generation: Following the procedure from (Nikolentzos & Vazirgiannis, 2020), we used five binary classification datasets consisting of random Erdős-Rényi (ER) graphs connected with motifs (positive example) or other smaller ER graphs with the same average degree as a motif (negative example), see Fig. 5 (details in the Appendix, Sec. A.2). For each dataset we constructed $S = 2048$ positive and S negative examples.

Tested Algorithms & Parameter Setting: We tested our GKAT, graph convolution networks (GCNs, Kipf & Welling, 2017), spectral graph convolution networks (SGCs, Defferrard et al., 2016) and graph attention networks (GATs, Velickovic et al., 2018). A feature vector in each vertex was of length $l = 5$ and contained top ordered l degrees of its neighbors (if there were fewer than l neighbors, we padded zeroes). A dataset for each motif was randomly split into $75\%/25\%$ training/validation set. We chose: the number of epochs $E = 500$, batch size $B = 128$, used Adam

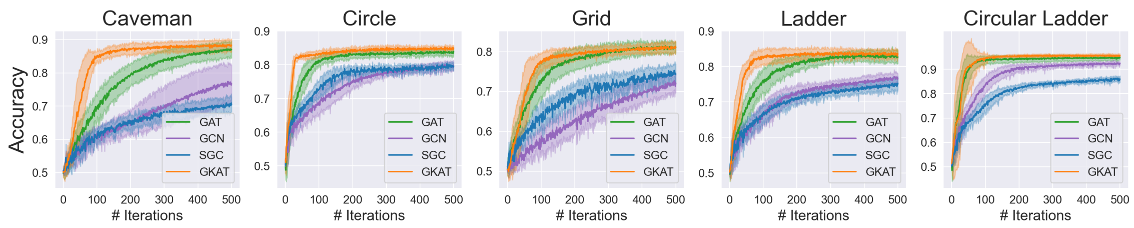

Figure 6. Model accuracy comparison of all four methods: GKAT, GAT, GCN and SGC on the motif-detection task. All architectures are 2-layer. GKAT outperforms other algorithms on all the tasks. See also Appendix:Sec. A.4 for the tabular version with $100K$-size graphs.

optimizer with learning rate $\eta = 0.001$ and early-stopped training if neither the validation loss nor validation accuracy improved for $c = 80$ continuous epochs.

We applied 2-layer architectures. For GCNs and SGCs, we used $h = 32$ nodes in the hidden layer. For SGC, we furthermore bound each hidden layer with 2 polynomial localized filters. For GAT and GKAT, we used 2 attention heads, with $h = 9$ nodes in the hidden layer to make all models of comparable sizes. In GKAT we used random walks of length $\tau = 3$. The results are presented in Fig. 6. GKAT outperforms all other methods for all the motifs.

5.1.2. Global graph properties & Deep vs Dense

Next we took as $\mathcal{H}$ an infinite family of motifs rather than just a single motif. The algorithm needs to decide whether a graph contains an induced cycle of length $> T$ for a given constant T. Thus the motif itself became a global property that cannot be detected by exploring just a close neighborhood of a node. In this experiment we focused also on the "depth versus density" trade-off. Shallow neural networks with dense attention are capable of modeling deeper networks relying on sparse layers, yet the price is extra computational cost per layer. We test here whether architectures that apply RWGNK kernels leveraging efficient decomposable long-range attention from Sec. 4.2 can also replace deeper counterparts or if they lose their expressiveness.

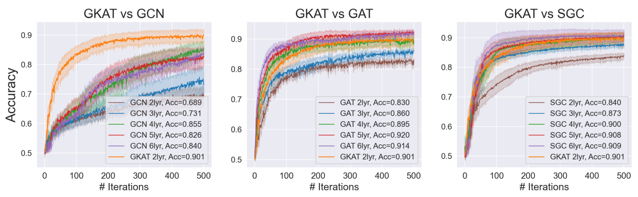

Figure 7. Comparison of the two-layer GKAT with different variants of GCNs, GATs and SGCs, varying by the number of hidden layers. Shallow GKAT architecture has the expressiveness of deeper version of its counterparts and in fact outperforms many of them (e.g. graph convolution networks.)

Dataset Generation: We created $S = 2048$ random binary trees, each having 50 nodes, with $75\%/25\%$ for training/validation. For each tree, we constructed a positive example, by connecting two nodes with the farthest distance from each other (a negative example was obtained by connecting two random vertices, but not farthest from each other). Note that a positive example constructed in such a way has shortest induced cycle of length $P + 1$, where P is the diameter of the tree.

Tested Algorithms & Parameter Setting: We used the same algorithms as before and run detailed ablation studies on the depth of the GKAT competitors, by comparing two-layer GKAT with GATs, GCNs and SGCs of up to six layers.

For a fair comparison, we used models with a comparable number of parameters. For the two-layer GKAT, we applied 8 heads in the first layer, and 1 head in the second layer. The dimension of each head was $d = 4$. The last layer was fully-connected with output dimensionality $o = 2$ for binary classification. We applied random walk length of $\tau = 6$. For GCN, GAT and SGC, we tested number of layers ranging from 2 to 6. We controlled the number of nodes in the hidden layer(s) for GCN, GAT and SGC, and the number of attention heads in each head for GAT so that their total number of trainable parameters was comparable with that of our two-layer GKAT. All other parameters were chosen as in Sec. 5.1.1. More details on parameter settings and additional ablation tests over random walk length of GKAT are given in Table 5 and Fig. 9 in the Appendix (Sec. A.2). Our main results are presented in Fig. 7.

We see that a shallow two-layer GKAT beats all GCN-variants (also deeper ones) as well as GATs and SGCs with < 4 layers by a wide margin. A two-layer GKAT is asymptotically equivalent to the four-layer GAT and SGC, yet as we show in Sec. 5.3, is faster to train and run inference on.

5.2. Bioinformatics & Social Networks experiments

Datasets: We tested GKAT for graph classification tasks on 9 standard and publicly available bioinformatics and social networks datasets (Kersting et al., 2016) using a carefully designed model selection and assessment framework for a fair comparison (Errica et al., 2020). The former include: D&D (Dobson & Doig, 2003), PROTEINS (Borgwardt et al., 2005), NCI1 (Wale et al., 2008) and ENZYMES (Schomburg et al., 2004), and the latter: IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-5K and COLLAB (Yanardag & Vishwanathan, 2015), see also Sec. A.3.1.

Tested Algorithms: We compared GKAT with top GNN methods used previously for that data: DCGNN (Zhang et al., 2018), DiffPool (Ying et al., 2018), ECC (Simonovsky & Komodakis, 2017), GraphSAGE (Hamilton et al., 2017) and RWNN (Nikolentzos & Vazirgiannis, 2020), which are selected based on their popularity and architectural differences. For bioinformatics datasets, but ENZYMES, we used the Molecular Fingerprinting (MF, Ralaivola et al., 2005; Luzhnica et al., 2019) as a baseline. This first applies global sum pooling and then a single-layer MLP with ReLU activations. For social datasets and ENZYMES, we applied the DeepMultisets (DM) method (Zaheer et al., 2017) as a baseline. We did not add the numbers for the GIN method (Xu et al., 2018) since we could not reproduce its reported results for the models of size similar to GKAT.

Table 1. Performance of different algorithms on the bioinformatics data. For each dataset, we highlighted/underlined the best/second best method. GKAT is the best on three out of four tasks.

	D&D	**NCI1**	**Proteins**	**Enzymes**
Baseline	<u>78.4 ±4.5%</u>	69.8±2.2%	**75.8±3.7%**	<u>65.2±6.4%</u>
DGCNN	76.6±4.3%	<u>76.4±1.7%</u>	72.9±3.5%	38.9±5.7%
DiffPool	75.0±3.5%	**76.9±1.9%**	73.7±3.5%	59.5±5.6%
ECC	72.6±4.1%	76.2±1.4%	72.3±3.4%	29.5±8.2%
GraphSAGE	72.9±2.0%	76.0±1.8%	73.0±4.5%	58.2±6.0%
RWNN	77.6±4.7%	71.4±1.8%	74.3±3.3%	56.7±5.2%
GKAT	**78.6±3.4%**	75.2±2.4%	**75.8 ±3.8%**	**69.7 ±6.0%**

Table 2. Performance of different algorithms on the social network data. GKAT is among two top methods for four out of five tasks.

	IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
Baseline	<u>70.8±5.0%</u>	**49.1 ±3.5%**	82.2±3.0%	52.2±1.5%	70.2±1.5%
DGCNN	69.2±5.0%	45.6±3.4%	87.8±2.5%	49.2±1.2%	71.2±1.9%
DiffPool	68.4±3.3%	45.6±3.4%	89.1±1.6%	<u>53.8±1.4%</u>	68.9±2.0%
ECC	67.7±2.8%	43.5±3.1%	OOM	OOM	OOM
GraphSAGE	68.8±4.5%	47.6±3.5%	84.3±1.9%	50.0±1.3%	**73.9±1.7%**
RWNN	<u>70.8±4.8%</u>	<u>47.8±3.8%</u>	**90.4±1.9%**	51.7±1.5%	71.7±2.1%
GKAT	**71.4±2.6%**	47.5±4.5%	<u>89.3±2.3%</u>	**55.3±1.6%**	<u>73.1±2.0%</u>

GKAT Setting: We used a two-layer GKAT followed by the baseline layers: we first applied an attention layer with k heads (a hyperparameter to be tuned), and then another one with one head to aggregate topological information on graphs. Next, we applied either the MF method or the DM method to further process the aggregated information. The random walk length τ in each GKAT layer satisfied $\tau \leq 4$ and depended on the evaluated datasets. The average graph diameter shown in Table 6 in the Appendix helps to calibrate walk length. We chose it to balance the pros of using a shallow architecture and the cons of information loss from dense layer compression. GKAT increased the number of the baseline's parameters by a negligible fraction.

Training Details: We used a 10-fold CV for model assessment, and an inner holdout with $90\%/10\%$ training/validation split for model selection following the same settings (Errica et al., 2020). We then trained the whole training-fold three times, randomly holding out 10% of data for early stopping after model selection in each fold. The average score for these runs was reported in Table 1 & 2.

The results from Table 1 and Table 2 show that GKAT is the best on three out of four bioinformatics datasets and is among two best methods on four out of five social network datasets. It is the only GNN method that consistently outperforms baseline on all but one bioinformatics dataset (bio-data benefits more than others from efficient longer-range attention modeling as showed by Choromanski et al. (2021)). In the Appendix (Sec. A.5) we provide additional comparisons of GKAT with GAT on citation networks, where GKAT outperforms GAT on two out of three tested datasets.

5.3. Space & Time Complexity Gains of GKAT

We measured speed and memory improvements coming from GKAT as compared to GAT as well as accuracy loss in comparison to Transformer using GKAT masking, but explicitly computed (GKAT-0), see Table 3. We decided to report relative rather than absolute numbers since the former are transferable across different computational setups. The accuracy gaps of the corresponding GKAT-0 and GKAT models (obtained after the same # of epochs) are marginal, yet GKAT yields consistent speed and memory gains as compared to GAT per attention layer, particularly substantial for very large graphs as those from Citeseer and Pubmed.

Table 3. Speed & Space Complexity gains provided by GKAT. **First row:** memory compression (lower better). **Second & third row:** speedup in training and inference respectively per one attention layer as compared to GAT. **Last row:** accuracy loss as compared to GKAT-0 applying brute-force RWGNK masking. We used four datasets from Sec. 5.1.1, a dataset from Sec. 5.1.2 (Tree) and two citation network datasets (see: Sec. A.5): Citeseer and Pubmed with graphs of much larger sizes and on which GKAT also outperforms GAT. We applied $r = 256$ random features to linearize softmax kernel for features in nodes for citation network datasets, $r = 16/8$ for datasets from Sec. 5.1.1/ 5.1.2.

	Cavem.	**Circle**	**Grid**	**Ladder**	**Tree**	**Citeseer**	**Pubmed**
GKAT / GKAT-0 memory	0.54	0.53	0.55	0.52	0.95	0.18	0.07
train speedup vs GAT	1.40x	1.41x	1.42x	1.40x	1.10x	5.10x	9.50x
inf speedup vs GAT	1.46x	1.49x	1.49x	1.47x	1.12x	5.21x	9.54x
GKAT-0 - GKAT (accur.)	0.07%	0.09%	0.08%	0.07%	0.06%	0.05%	0.06%

In Table 4 we show that GKAT is also faster that its counterparts (GCN, GAT, SGC) in terms of wall clock time needed to reach particular accuracy levels, by comparing accuracy levels reached by different models in a given wall clock time budget (time GKAT needs to complete first 100 epochs).

Table 4. Running time of training different networks on datasets from Sec. 5.1.1 and Sec. 5.1.2. For GCN, GAT and SGC, we reported the accuracy with 2 layers. For GKAT, we used a 2-layer architecture and reported the accuracy with a fixed walk length of 6 for Induced Cycle Detection, and of 3 for motifs from Sec. 5.1.1.

	Induced Cycle	**Caveman**	**Circle**	**Grid**	**Ladder**	**Circle Ladder**
GCN	63.2%	62.1%	71.4%	59.3%	66.7%	87.4%
GAT	77.0%	69, 1%	80.6%	73.8%	75.9%	93.7%
SGC	56.6%	55.4%	64.7%	58.2%	59.1%	66.5%
GKAT	**83.6%**	**85.1%**	**83.3%**	**77.1%**	**82.4%**	**94.6%**

5.4. 2-level block-Toeplitz masks for vision data

In this section (see: Fig. 8), we present additional results in the vision domain, showing large, **+2.5-3.4** percentage point, gains in accuracy arising from applying the 2-level block Toeplitz masking introduced in the paper (see: Sec. 3.2) on top of the regular vision Performer. As explained before, the price we pay for these gains is only a $\log(L)$ multiplicative factor in time complexity.

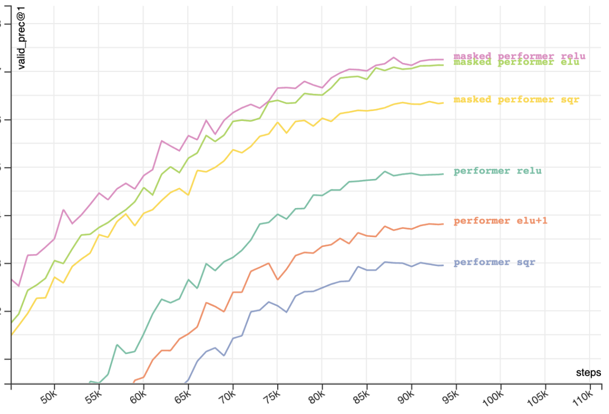

Figure 8. Comparison of regular Performers using x^2, ELU $+ 1$, and ReLU kernels, with their counterparts applying 2-level block Toeplitz masking from our paper on the ImageNet classification task (hidden size = 768, 12 layers & heads, MLP dim = 3072).

6. Additional results & some open questions

In this section, we present additional theoretical results regarding the theory of the scalable efficient masked Transformers that we have developed. We focus on masking mechanisms for the tree-graph inputs. We also discuss some open problems for future work.

In Section 3.3 we showed that for the specific classes of functions f (exponentials of affine mappings of the shortest-distance paths) and arbitrary weighted trees, pairs (G_{base}, f) are tractable. Here we will provide additional related results, but for arbitrary functions f. Our first result is as follows:

Lemma 6.1. *If $\mathcal{T} = G_{\mathrm{base}}$ is an unweighted tree and f is an arbitrary function then the corresponding mask matrix $\mathbf{M} = \mathbf{M}(G_{\mathrm{base}}, f)$ supports matrix-vector multiplication in time $O(L \cdot \log^2(L))$. Thus (G_{base}, f) is tractable.*

We now show that if the diameter $\mathrm{diam}(\mathcal{T})$ of the tree $\mathcal{T}$ is of the order of magnitude $o(\log^2(L))$, where $L = |V(\mathcal{T})|$, a more efficient algorithm can be used.

Lemma 6.2. *If $\mathcal{T} = G_{\mathrm{base}}$ is an unweighted tree and f is an arbitrary function then the corresponding mask matrix $\mathbf{M} = \mathbf{M}(G_{\mathrm{base}}, f)$ supports matrix-vector multiplication in time $O(L \cdot \mathrm{diam}(G_{\mathrm{base}}))$. Thus (G_{base}, f) is tractable.*

Corollary 6.3. *From the above, we obtain an $O(L\log(L))$ algorithm for computing the action of $\mathbf{M}$ on $\mathbf{x}$ if G_{base} is a tree where only a constant number of its vertices that are not leaves are of degree at most two (e.g. if G_{base} is a d-regular tree for $d \geq 3$ or if G_{base} is a binary tree).*

Corollary 6.4. *The algorithm from the proof of Lemma 6.2 (see:Appendix) can be used to improve algorithm from Lemma 6.1 (that works for graphs with arbitrary diameters) if the input $\mathcal{T}$ to that algorithm satisfies: $\mathrm{diam}(\mathcal{T}) = o(\log^2(|V(\mathcal{T})|))$. Note that computing the diameter of any tree $\mathcal{T}$ can be done in time $O(|V(\mathcal{T})|)$ by running two depth-first-search procedures: the first one from an arbitrary nodes v of $\mathcal{T}$ and the second one from the node farthest from v in $\mathcal{T}$ (obtained via the first depth-first-search procedure).*

We leave the Reader with an interesting open problem:

Can we improve Lemma 6.1 to obtain $O(L\log(L))$ running time, i.e. replace the $\log^2(L)$ factor with a $\log(L)$ factor?

Note that if the unweighted tree is a single path, the answer to the above question is: Yes. Indeed, this is precisely the 1D-RPE setting that we have discussed before. Furthermore, since in that setting the problem reduced to the multiplication with Toeplitz matrices, inherently relying on the FFT, the $\log(L)$ factor in all likelihood cannot be improved (unless FFT can be replaced with a faster algorithm or multiplication with Toeplitz matrices is conducted approximately). Still, we do not know whether for general unweighted trees (or even nontrivial tree-extensions of the path) we can reach the $O(L\log(L))$ running time.

It might be also interesting to analyze how those of our presented methods that work for tree input data can be extended to non-tree graphs, but with low treewidth (Cygan et al., 2015), that can be thought of as relaxations of trees.

7. Conclusion

We presented a holistic approach to incorporating masking into scalable low-rank Transformers. We provided general theoretical results which include earlier results as special cases. We conducted comprehensive empirical evaluations of the new instantiations of the mechanism for graph data.

We focused in the paper not only on scalable variants, but have introduced several new masking meethods that can be used on their own, even in regular Transformers. These include in particular d-level block-Toeplitz masking mechanisms with applications in vision and video processing, that we believe might lead to new vision-Transformers architectures. We show that topological masking is a powerful inductive bias and that corresponding “topological Transformers” turn out to be effective in various domains such as bioinformatics and vision.

8. Acknowledgements

AW acknowledges support from a Turing AI Fellowship under grant EP/V025279/1, The Alan Turing Institute, and the Leverhulme Trust via CFI.

References

Al-Mohy, A. H. and Higham, N. J. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.*, 33(2):488–511, 2011. doi: 10.1137/100788860. URL https://doi.org/10.1137/100788860.

Artemova, S., Grudinin, S., and Redon, S. Fast construction of assembly trees for molecular graphs. *J. Comput. Chem.*, 32(8):1589–1598, 2011. doi: 10.1002/jcc.21738. URL https://doi.org/10.1002/jcc.21738.

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., Assael, Y., Jumper, J., Kohli, P., and Kelley, D. R. Effective gene expression prediction from sequence by integrating long-range interactions. *bioRxiv*, 2021. doi: 10.1101/2021.04.07.438649. URL https://www.biorxiv.org/content/early/2021/04/08/2021.04.07.438649.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21 (suppl_1):i47–i56, 06 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti1007. URL https://doi.org/10.1093/bioinformatics/bti1007.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Belanger, D., Colwell, L., and Weller, A. Masked language modeling for proteins via linearly scalable long-context transformers. *arXiv preprint arXiv:2006.03555*, 2020.

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

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms*. Springer, 2015. ISBN 978-3-319-21274-6. doi: 10.1007/978-3-319-21275-3. URL https://doi.org/10.1007/978-3-319-21275-3.

Dai, Z., Lai, G., Yang, Y., and Le, Q. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Daniely, A., Frostig, R., Gupta, V., and Singer, Y. Random features for compositional kernels. *CoRR*, abs/1703.07872, 2017. URL http://arxiv.org/abs/1703.07872.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016. URL http://arxiv.org/abs/1606.09375.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771—783, July 2003. ISSN 0022-2836. doi: 10.1016/s0022-2836(03)00628-4. URL https://doi.org/10.1016/s0022-2836(03)00628-4.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Errica, F., Podda, M., , Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification. *ICLR 2020*, 2020.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y., and Tao, D. A survey on visual transformer. *CoRR*, abs/2012.12556, 2020. URL https://arxiv.org/abs/2012.12556.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. *URL https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets*, 795, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Kondor, R. and Lafferty, J. D. Diffusion kernels on graphs and other discrete input spaces. In Sammut, C. and Hoffmann, A. G. (eds.), *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pp. 315–322. Morgan Kaufmann, 2002.

Lee, D. Fast multiplication of a recursive block toeplitz matrix by a vector and its application. *J. Complex.*, 2(4):295–305, 1986. doi: 10.1016/0885-064X(86)90007-5. URL https://doi.org/10.1016/0885-064X(86)90007-5.

Li, D., Zhang, X., and Liu, R. Exponential integrators for large-scale stiff Riccati differential equations. *J. Comput. Appl. Math.*, 389:113360, 2021. doi: 10.1016/j.cam.2020.113360. URL https://doi.org/10.1016/j.cam.2020.113360.

Liutkus, A., Cífka, O., Wu, S., Simsekli, U., Yang, Y., and Richard, G. Relative positional encoding for transformers with linear complexity. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7067–7079. PMLR, 2021. URL http://proceedings.mlr.press/v139/liutkus21a.html.

Luo, S., Li, S., Cai, T., He, D., Peng, D., Zheng, S., Ke, G., Wang, L., and Liu, T. Stable, fast and accurate: Kernelized attention with relative positional encoding. *CoRR*, abs/2106.12566, 2021. URL https://arxiv.org/abs/2106.12566.

Luzhnica, E., Day, B., and Liò, P. On graph classification networks, datasets and baselines. *arXiv preprint arXiv:1905.04682*, 2019.

Nikolentzos, G. and Vazirgiannis, M. Random walk graph neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 16211–16222. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ba95d78a7c942571185308775a97a3a0-Paper.pdf.

Orecchia, L., Sachdeva, S., and Vishnoi, N. K. Approximating the exponential, the lanczos method and an õ(m)-time spectral algorithm for balanced separator. In Karloff, H. J. and Pitassi, T. (eds.), *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pp. 1141–1160. ACM, 2012. doi: 10.1145/2213977.2214080. URL https://doi.org/10.1145/2213977.2214080.

Parmar, N., Ramachandran, P., Vaswani, A., Bello, I., Levskaya, A., and Shlens, J. Stand-alone self-attention in vision models. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 68–80, 2019.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. In

9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=QtTKTdVrFBB.

Pham, N., Ha, T., Nguyen, T., Nguyen, T., Salesky, E., Stüker, S., Niehues, J., and Waibel, A. Relative positional encoding for speech recognition and direct translation. In Meng, H., Xu, B., and Zheng, T. F. (eds.), *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pp. 31–35. ISCA, 2020. doi: 10.21437/Interspeech.2020-2526. URL https://doi.org/10.21437/Interspeech.2020-2526.

Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., and Zhong, Y. cosformer: Rethinking softmax in attention. *CoRR*, abs/2202.08791, 2022. URL https://arxiv.org/abs/2202.08791.

Ralaivola, L., Swamidass, S. J., Saigo, H., and Baldi, P. Graph kernels for chemical informatics. *Neural networks*, 18(8):1093–1110, 2005.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Trans. Assoc. Comput. Linguistics*, 9:53–68, 2021. URL https://transacl.org/ojs/index.php/tacl/article/view/2405.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(Database issue):D431–3, January 2004.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. Collective classification in network data, 2008.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. In Walker, M. A., Ji, H., and Stent, A. (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pp. 464–468. Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-2074. URL https://doi.org/10.18653/v1/n18-2074.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3693–3702, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Vaswani, A., Ramachandran, P., Srinivas, A., Parmar, N., Hechtman, B. A., and Shlens, J. Scaling local self-attention for parameter efficient visual backbones. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 12894–12904. Computer Vision Foundation / IEEE, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Vaswani_Scaling_Local_Self-Attention_for_Parameter_Efficient_Visual_Backbones_CVPR_2021_paper.html.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, 2008. doi: 10.1007/s10115-007-0103-5. URL https://doi.org/10.1007/s10115-007-0103-5.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 14138–14148. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17664.

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

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783417. URL https://doi.org/10.1145/2783258.2783417.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 5754–5764, 2019.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T. Do transformers really perform bad for graph representation? *CoRR*, abs/2106.05234, 2021a. URL https://arxiv.org/abs/2106.05234.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T. Do transformers really perform bad for graph representation? *CoRR*, abs/2106.05234, 2021b. URL https://arxiv.org/abs/2106.05234.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/e77dbaf6759253c7c6d0efc5690369c7-Paper.pdf.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification, 2018. URL https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146.

Zhou, P., Fan, R., Chen, W., and Jia, J. Improving generalization of transformer for speech recognition with parallel schedule sampling and relative positional embedding. *CoRR*, abs/1911.00203, 2019. URL http://arxiv.org/abs/1911.00203.

A. Appendix

A.1. Several Pointers

We include pointers to this part of the code that does not include sensitive/proprietary information. The core GKAT framework (with the additional analysis of new graph sketches that GKAT leads to, called *graphots*, mixing regular feature vectors in nodes with the topological features) is here: https://github.com/HL-hanlin/GKAT. We used (deterministic and random) feature map mechanisms corresponding to the features defined in graph nodes from this repository: https://github.com/google-research/google-research/tree/master/performer.

A.2. Combinatorial Classification Experiments: Additional Details

The data for the motif detection task from Section 5.1.1 was generated as follows:

- Firstly we created five simple motifs as shown in Fig. 5. Note that each motif has ≥ 9 vertices so a brute-force combinatorial algorithm for motif-detection would take time $\Omega(N^9)$, prohibitively expensive even for small graphs G.
- We then generated for each motif S small Erdős-Rényi graphs with the same number of nodes as that motif and the same average degree.
- For each motif, we also generated S larger Erdős-Rényi random graphs, each of 100 vertices, again of the same average degree.
- We obtained positive/negative samples by connecting each larger Erdős-Rényi random graph with the motif/previously generated smaller Erdős-Rényi random graph (with certain edge probability).

In Table 5 we present additional details regarding architectures used in the experiments from Section 5.1.2, in particular the number of parameters and heads / polynomial filters used in different layers. Ablation tests over GKAT random walk length for Section 5.1.2 are presented in Fig. 9.

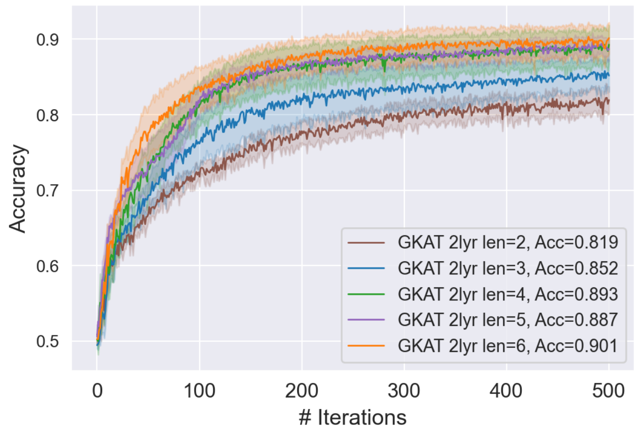

Figure 9. Ablation tests over random walk length of GKAT in Sec. 5.1.2.

A.3. GNNs for Bioinformatics Tasks & Social Networks Data: Additional Details

A.3.1. Datasets Descriptions

Detailed profiles of the datasets used in the experiments from Sec. 5.2 are given in Table 6.

For each dataset, we chose graphs with the number of nodes close to the average number of nodes shown in Table 6. Examples of bioinformatics-graphs from these datasets are given in Fig. 10. Examples of social network graphs from these datasets are given in Fig. 11.

A.3.2. Hyperparameter Selection

In this section, we present details regarding hyperparameter selection in Section 5.2 (see: Table 7).

Table 5. Additional details regarding architectures used in Section 5.1.2. For GKAT, we applied 8 heads in the first layer, and 1 head in the second layer, with 4 hidden units in each attention head. The total number of trainable parameters was 242. For GAT, we tested the number of layers from 2 to 6, changing the number of attention heads in each layer, but with the same number of hidden units in each attention head. For GCN, we modified the number of hidden units in each layer. For SGC, we modified the number of polynomial filters and the number of hidden units in each layer. The number of attention heads in GAT, as well as the number of hidden units in each layer in GCN and SGC were chosen to make their total number of trainable parameters comparable with the corresponding number of GKAT.

		#Heads	**Dim. Head**	**#Parameters**
GKAT	**2 layers**	$[8,1]$	4	242
	2 layers	$[8,1]$	4	242
GAT	**3 layers**	$[4,2,1]$	4	242
	4 layers	$[4,2,1,1]$	4	266
	5 layers	$[2,2,2,1,1]$	4	258
	6 layers	$[3,2,1,1,1,1]$	4	270
			Dim. Layer	**#Parameters**
	2 layers		$[14,14]$	268
GCN	**3 layers**		$[10,10,10]$	262
	4 layers		$[8,8,8,8]$	250
	5 layers		$[10,8,6,6,6]$	260
	6 layers		$[10,6,6,6,6,6]$	268
		#Polynomial Filters	**Dim. Layer**	**#Parameters**
	2 layers	$[4,2]$	$[10,8]$	236
SGC	**3 layers**	$[4,2,2]$	$[8,6,6]$	234
	4 layers	$[8,2,2,2]$	$[8,5,4,4]$	247
	5 layers	$[8,2,2,2,2]$	$[6,5,4,4,4]$	245
	6 layers	$[8,2,2,2,2,2]$	$[6,4,4,4,4,3]$	249

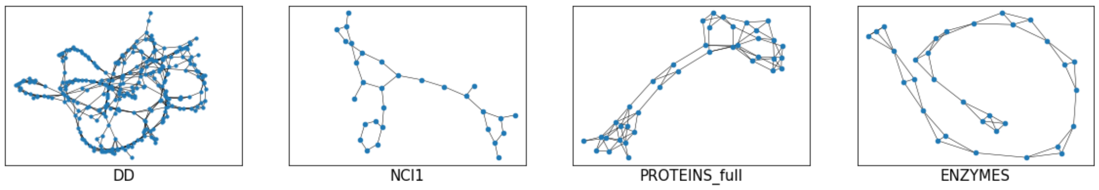

Figure 10. Representative plots for bioinformatics datasets. For each bioinformatics dataset, we chose the graph with number of nodes most similar to the average number of nodes shown in Table 6.

The tunable parameters included: general parameters like batch size, learning rate, dropout ratio, global pooling methods, regularization rate, data normalization methods, as well as parameters specific to our GKAT layers, which included: number of GKAT layers, number of attention heads and dimension of each head in a GKAT layer. We also tuned other options: whether to add a fully-connected layer after data normalization, but before GKAT layers, and dimension of fully-connected layers (both preceding and coming after GKAT layers). Due to the large amount of tunable parameters, we decided to first conduct a rough search for each parameter using only one random CV fold, select one/several parameter combination(s) with best performance, and then reused on all other folds.

For all other methods, we reported the best scores conducted via an extensive hyperparameters grid search (Errica et al., 2020). For GKAT, we fixed the number of epochs to $E = 1000$, early stopping patience as 500 epochs, the criterion for early stopping as validation accuracy, global pooling method as summation, and used Adam optimizer. Then we performed hyperparameter tuning for: batch size $B \in \{32, 128\}$, learning rate $\eta \in \{0.01, 0.001, 0.0001\}$, dropout ratio $\in \{0.0, 0.1, 0.2, 0.4\}$, L_2-regularization rate $\in \{0.001, 0.005\}$, dimension of attention head in the first GKAT layer $h \in \{4, 8, 16, 32\}$, number of attention heads in the first GKAT layer $\in \{1, 4, 8, 12\}$, number of nodes in the MLP layer $\in \{32, 64, 128\}$, GKAT random walk length $\tau \in \{1, 2, 3, 4\}$, whether to use a fully-connected layer before the first GKAT layer, and whether to apply batch normalization to pre-prosess data before feeding the model with it.

For some of the datasets (e.g. D&D), we selected the best hyperparameter set optimized over one random CV fold, and used it across all cross-validation outer folds.

Table 6. Bioinformatics and Social Dataset descriptions. #NODES, #EDGES and $\mathrm{Diameter}$ columns contain values averaged over all graphs in a given dataset.

		#Graphs	#Classes	#Nodes	#Edges	Diameter	#Features
BIOINF.	D&D	1178	2	284.32	715.66	19.90	89
	ENZYMES	600	6	32.63	64.14	10.86	3
	NCI1	4110	2	29.87	32.30	13.26	37
	PROTEINS	1113	2	39.06	72.82	11.48	3
SOCIAL	COLLAB	5000	3	74.49	2457.78	1.86	1
	IMDB-BINARY	1000	2	19.77	96.53	1.86	1
	IMDB-MULTI	1500	3	13.00	65.94	1.47	1
	REDDIT-BINARY	2000	2	429.63	497.75	9.72	1
	REDDIT-5K	4999	5	508.82	594.87	11.96	1

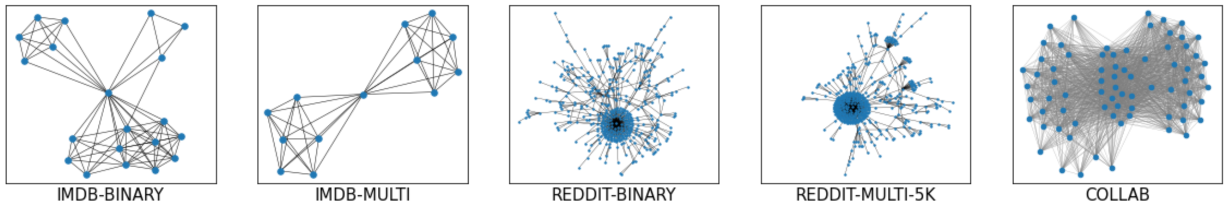

Figure 11. Representative plots for social datasets. For each social dataset, we chose the graph with number of nodes most similar to the average number of nodes shown in Table 6.

A.4. Space & Time Complexity Gains of GKAT: Additional Experiments

Additionally, we have conducted experiments on much larger Erdős-Rényi graphs with motifs, see: Section 5.1.1. The average number of nodes of each ER graph was 100K. Tested architectures had the same characteristics as in Section 5.1.1. The results (final model accuracy) are presented in Table 8 for datasets: Caveman, Circle, Grid, Ladder and Circular-Ladder respectively:

A.5. Experiments with Citation Networks Datasets

A.5.1. Datasets Descriptions

Datasets: To directly compare GKAT with GAT, we also tested both algorithms on three publicly available citation networks datasets: Cora, Citeseer and Pubmed ((Sen et al., 2008)) with the same data splits as in (Velickovic et al., 2018). Datasets descriptions are given in Table 9.

A.5.2. Comparison with GAT

Experiment Settings: We used the same model architecture and parameters as in GAT for our GKAT to make the comparison as accurate as possible. The only difference is that we replaced the adjacency matrix masking in GAT by the normalized dot-product based similarity matrix generated from random walks, as described in Section 4.2. Both models used two-layer attention, with 8 attention heads in the first layer, and 1 head in the second layer. We used 8 hidden units in the first layer, and the number of output units in the second layer was the same as number of classes. Each layer was followed by an exponential linear unit (ELU) activation. We applied L_2-regularization with $\lambda = 0.0005$, dropout with $p = 0.6$ for inputs and normalized attention coefficients in both layers for all three datasets.

Results: The results are shown in Table 10. Our GKAT algorithm achieved lower accuracy on Cora dataset, but higher on the remaining two.

Dynamic Generator of Random Walks: We also tried the so-called *dynamic*-GKAT. The dynamic variant generated random walks from scratch in each training epoch, thus requiring additional compute. However one advantage of the dynamic version is that we could assign different transition probabilities for adjacent nodes (rather than sampling next

Table 7. Hyperparameter settings for the bioinformatics and social network datasets from Section 5.2.

		BS	#Heads	d_{Head}	d_{FC}	Len_{rw}	Drop	L2	add FC	Norm
CHEM.	**D&D**	32	8	16	128	2	–	0.005	No	BN
	NCI1	32	8	32	64	4	0.1	0.001	Yes	BN
	PROTEINS	32 128	4 8	8 32	32 128	3	–	0.001	Yes	BN
	ENZYMES	32	4 8	16 32	32 64	3	0.1	0.001	Yes	BN
SOCIAL	**COLLAB**	32	12	4	128	2	–	0.005	No	No
	IMDB-BINARY	32	8 12	4 8	64 128	1	–	0.005	No	No BN
	IMDB-MULTI	32	8 12	4 8	64 128	1	–	0.005	No	No BN
	REDDIT-BINARY	32	4	4	128	2	–	0.005	No	BN
	REDDIT-5K	32	8	8	64	2	–	0.005	No	BN

Table 8. Running time of training different networks on datasets from Sec 5.1.1 but with much larger number of nodes ($\sim$ 100K). For GCN, GAT and SGC, we reported the accuracy with 2 layers. For GKAT, we used a 2-layer architecture and reported the accuracy with a fixed walk length of 3 for motifs from Sec. 5.1.1.

	Caveman	Circle	Grid	Ladder	Circle Ladder
GCN	88.3%	82.7%	80.4%	80.6%	91.4%
GAT	75.0%	81.0%	69.8%	77.0%	89.2%
SGC	70.0%	80.4%	72.3%	76.1%	82.4%
GKAT	**89.3%**	**83.2%**	**80.7%**	**81.5%**	**92.3%**

Table 9. Citation Networks Datasets Descriptions.

	Cora	Citeseer	Pubmed
#Nodes	2708	3327	19717
#Edges	5419	4732	44338
#Features	1433	3703	500
#Classes	7	6	3
#Training Nodes	140	120	60
#Validation Nodes	500	500	500
#Test Nodes	1000	1000	1000

Table 10. Comparison of GAT and GKAT on citation networks datasets. For Cora and Citeseer, we reported the results for GAT from (Velickovic et al., 2018). For GAT and Pubmed dataset, we reported the results averaged over 15 runs with the same parameter settings as in Cora and Citeseer. GKAT was run 15 times over multiple random walk lengths up to 7, and the best was reported.

	Cora	Citeseer	Pubmed
GAT	**83.0±0.7%**	$72.5 \pm 0.7\%$	$77.2 \pm 0.6\%$
GKAT	$82.1 \pm 0.7\%$	**73.0±0.7%**	**78.0 ±0.7%**

point of the walk uniformly at random). The transition probability matrix can be a masked attention matrix and we only need its actions on the L-dimensional vectors to compute probability vectors in the visited nodes. Since the GKAT-masked attention can be interpreted as a kernelizable attention of a product-kernel (the product of the kernel between feature vectors in nodes and the nodes in the graph) and each factor-kernel admits on expectation a linearization, the product-kernel also does it via the mechanism of the Cartesian-product random features (see: (Daniely et al., 2017)). Thus this matrix admits on expectation a lower-rank decomposition and thus based on our analysis from Sec. 3.4, the actions of that matrix on the input vectors can be efficiently computed.

An intuition behind that particular variant is that we assign higher transition probabilities for neighbors with higher attention coefficients. The dynamic variant enabled us to improve accuracy of GKAT on Citeseer to **73.3**% (with reduced **0.6**% standard deviation).

A.5.3. Ablation Tests on Random Walk Length for GKAT

Figure 12 compares the effect of random walk path length of GKAT algorithms on training for Cora, Citeseer and Pubmed datasets. We run GKAT with multiple random walk lengths up to 7. The results show that a small path length no longer than 4 is enough for GKAT and dynamic-GKAT, which supports our claim that short walks are sufficient for GKAT.

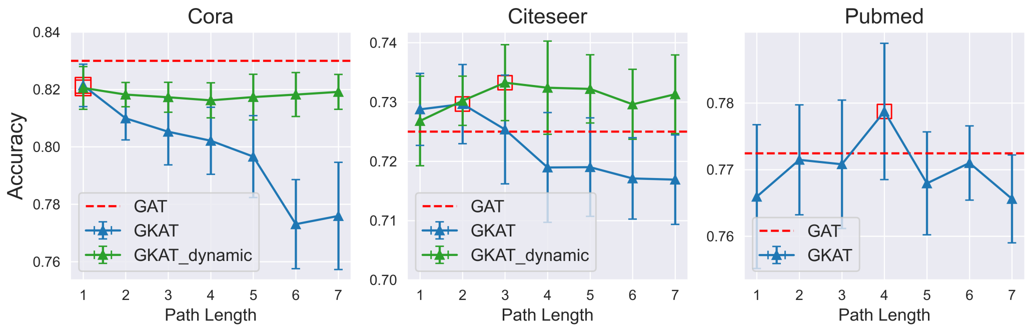

Figure 12. Ablation tests over random walk path lengths of GKAT and dynamic-GKAT on Cora, Citeseer and Pubmed datasets. The errorbar represents 1 standard deviation. Test-accuracies for the optimized GAT were shown as horizontal red dotted lines in each subplot. Best test accuracies of GKAT algorithms were highlighted with red squares. The dynamic-GKAT was tested only on datasets, where accuracy advantage of the regular optimized GKAT over optimized GAT was $\leq 0.5\%$.

A.6. Proof of Lemma 3.1 and Algorithm 1

Proof. Note that the ith token representation $\mathbf{r}_i$ obtained from the general masked kernel attention is of the following form:

$$\mathbf{r}_i = \frac{\phi(\mathbf{q}_i^\top)^\top \sum_{j=1}^{L} \mathbf{M}_{i,j}\phi(\mathbf{k}_j^\top)\mathbf{v}_j}{\phi(\mathbf{q}_i^\top)^\top \sum_{j=1}^{L} \mathbf{M}_{i,j}\phi(\mathbf{k}_j^\top)}, \tag{7}$$

where $\mathbf{q}_i, \mathbf{k}_j, \mathbf{v}_j$ stand for the ith query and jth key/value row vectors respectively. As in (Luo et al., 2021), we define the following two sequences of matrices and vectors of shapes $\mathbf{R}^{m\times d}$ and $\mathbf{R}^{1\times m}$ respectively:

$$\mathbf{D}^1 = \left(\sum_{j=1}^{L} \mathbf{M}_{i,j}\phi(\mathbf{k}_j^\top)\mathbf{v}_j\right)_{i=1}^{L}, \ \mathbf{D}^2 = \left(\sum_{j=1}^{L} \mathbf{M}_{i,j}\phi(\mathbf{k}_j^\top)^\top\right)_{i=1}^{L}. \tag{8}$$

If we define $\tilde{\mathbf{D}}^1$ and $\tilde{\mathbf{D}}^2$ as the vectorized variants of $\mathbf{D}_1$ and $\mathbf{D}_2$, where each element of the sequence is (row) vectorized (note that the elements of $\mathbf{D}^2$ are already row-vectorized) and the resulting vectors are stacked into matrices, then:

$$\tilde{\mathbf{D}}^1 = \mathbf{M}\mathbf{V}^1, \ \tilde{\mathbf{D}}^2 = \mathbf{M}\mathbf{V}^2, \tag{9}$$

with the ith rows of $\mathbf{V}^1$ and $\mathbf{V}^2$ given as: $\mathbf{V}^1_i = \mathrm{vec}(\phi(\mathbf{k}_i)^\top \mathbf{v}_i)$, $\mathbf{V}^2_i = \phi(\mathbf{k}_i^\top)^\top$.

We conclude that the computation of $\mathbf{D}_1$ and $\mathbf{D}_2$ takes time $T_{\mathbf{N}}(L)md$ and consequently all representations $\mathbf{r}_i$ can be computed in time $O((T_{\mathbf{N}}(L) + L)md)$. That completes the proof of Lemma 3.1. □

Note that Algorithm 1 follows immediately from the above proof. The algorithm consists of two phases. In the first phase, sequences $\mathbf{D}^1$ and $\mathbf{D}^2$ are computed (in the form of $\tilde{\mathbf{D}}^1$ and $\tilde{\mathbf{D}}^2$). In the second phase, they are used to get new token-representations (different tokens apply different elements of $\mathbf{D}^1$ and $\mathbf{D}^2$ for the computation of their new representations).

A.7. Proof of Lemma 3.3

Proof. We define the so-called *grid-ordering* on the nodes of the d-dimensional grid recursively. For $d = 1$ we take the natural ordering on the line. For the d-dimensional grid with $d > 1$, we take the $(d-1)$-dimensional slices: $\mathcal{S}_1, \mathcal{S}_2, ...$ ordered according to their dth-dimension index. In each slice we recursively order all the tokens We then combine the orderings of all the slices to get the ordering on the d-dimensional grid.

We claim that the mask corresponding to the d-dimensional grid with the grid-ordering of tokens is d-level block-Toeplitz.

We will proceed by an induction on d. For $d = 1$ the corresponding mask is of the form: $\mathbf{M} = [f(i-j)]_{i,j=1,...,L}$ for some learnable function f and a natural ordering on the line. Therefore $\mathbf{M}$ is constant on each diagonal thus it is Toeplitz (e.g. 1-level block-Toeplitz). Now let us assume that $d > 1$ and the result holds for $d - 1$. We take the grid-ordering for the d-dimensional grid and the partitioning of the tokens given by the $(d-1)$-dimensional slices $\mathcal{S}_1, \mathcal{S}_2, ...$ ordered according to their dth-dimension index. This partitioning induces the block-partitioning of the mask $\mathbf{M}$ (using grid-ordering) on the d-dimensional grid. Now note that the shortest-path distance between two nodes v_1 and v_2 from slices $\mathcal{S}_{i_1}$ and $\mathcal{S}_{i_2}$ respectively can be computed as: $\mathrm{dist}(v_1, v_2') + |i_1 - i_2|$, where v_2' is the projection of v_2 into slice $\mathcal{S}_{i_1}$. This observation combined with the inductive assumption implies that defined above block-partitioning of $\mathbf{M}$ produces matrices $\mathbf{B}^{i,j}$ from Definition 3.2 and that completes the proof. □

A.8. Proof of Lemma 3.5

Proof. Without loss of generality we can assume that G_{base} is a tree. Assume that τ is of the form: $\tau(z) = az + b$ for some $a, b \in \mathbb{R}$. Take some $\mathbf{x} \in \mathbb{R}^L$. Let us root the tree $\mathcal{T}$ in one of its vertices that we denote as v_0. Denote by $v_1, ..., v_k$ for some $k \geq 0$ its neighbors. For every node i we define s_i as follows:

$$s_i = \sum_{j \in \mathcal{T}_i} \exp(\tau(\mathrm{dist}(i,j)))\mathbf{x}_j, \tag{10}$$

where $\mathcal{T}_i$ denotes a subtree of $\mathcal{T}$ rooted in i. Our first observation is that all s_i for $i = 1, ..., L$ can be computed in $O(L)$ time. To see this, note that:

$$\begin{gathered} s_{v_0} = \exp(\tau(\mathrm{dist}(v_0, v_0)))\mathbf{x}_0 + \sum_{l=1,\ldots,k} \sum_{j \in \mathcal{T}_{v_l}} \exp(\tau(\mathrm{dist}(v_0, j)))\mathbf{x}_j = \\ e^b\mathbf{x}_0 + \sum_{l=1,\ldots,k} \sum_{j \in \mathcal{T}_{v_l}} e^{a \cdot \mathrm{dist}(v_0,j)+b}\mathbf{x}_j = e^b\mathbf{x}_0 + \sum_{l=1,\ldots,k} \sum_{j \in \mathcal{T}_{v_l}} e^{a \cdot (\mathrm{dist}(v_l,j)+W(v_0,v_l))+b}\mathbf{x}_j = \\ e^b\mathbf{x}_0 + \sum_{l=1,\ldots,k} e^{W(v_0,v_l)} \sum_{j \in \mathcal{T}_{v_l}} e^{a \cdot \mathrm{dist}(v_l,j)+b}\mathbf{x}_j = e^b\mathbf{x}_0 + \sum_{l=1,\ldots,k} e^{W(v_0,v_l)} s_{v_l} \end{gathered} \tag{11}$$

Thus we see that computing s_{v_0} requires computing each s_{v_l}, followed by additional addition/multiplication operations that take time $O(\deg(v_0))$. We conclude that we can recursively compute all s_i in time $O(L)$. Let us note that the entry of $\mathbf{w} = \mathbf{M}\mathbf{x}$ corresponding to node i is of the form:

$$\mathbf{w}_i = \sum_{j \in \mathcal{T}} \exp(\tau(\mathrm{dist}(i,j)))\mathbf{x}_j, \tag{12}$$

Therefore ultimately we aim to compute all $\mathbf{w}_i$ for $i = 1, ..., L$ in time $O(L)$. We observe that:

$$\mathbf{w}_{v_0} = s_{v_0} \tag{13}$$

Now take node $i \neq v_0$. Denote by $p(i)$ the predecessor of i in $\mathcal{T}$. We have:

$$\begin{gathered} \mathbf{w}_i = \sum_{j \in \mathcal{T}_i} \exp(\tau(\mathrm{dist}(i,j)))\mathbf{x}_j + \sum_{j \notin \mathcal{T}_i} \exp(\tau(\mathrm{dist}(i,j)))\mathbf{x}_j = \\ s_i + \sum_{j \notin \mathcal{T}_i} e^{a \cdot \mathrm{dist}(i,j)+b}\mathbf{x}_j = s_i + \sum_{j \notin \mathcal{T}_i} e^{a(W(i,p(i))+\mathrm{dist}(p(i),j))+b}\mathbf{x}_j = \\ s_i + e^{aW(i,p(i))} \sum_{j \notin \mathcal{T}_i} e^{a\mathrm{dist}(p(i),j)+b}\mathbf{x}_j = s_i + e^{aW(i,p(i))} t_i, \end{gathered} \tag{14}$$

where $t_i = \sum_{j \notin \mathcal{T}_i} e^{a\mathrm{dist}(p(i),j)+b}\mathbf{x}_j$. Now note that:

$$\mathbf{w}_{p(i)} = t_i + e^{W(i,p(i))}s_i \tag{15}$$

and thus: $t_i = \mathbf{w}_{p(i)} - e^{W(i,p(i))}s_i$. Plugging in the formula for t_i into Equation 14, we get:

$$\mathbf{w}_i = e^{W(i,p(i))}\mathbf{w}_{p(i)} + (1 - e^{2W(i,p(i))})s_i \tag{16}$$

We conclude that having computed all s_i for $i = 1, ..., L$ in time $O(L)$, we can compute all $\mathbf{w}_i$ for $i = 1, ..., L$ in time $O(L)$ by ordering vertices in their increasing distance from the root v_0, setting up $\mathbf{w}_{v_0} = s_{v_0}$ and applying Equation 16. $\square$

A.9. Proof of Theorem 4.1

Proof. We need the following definition.

Definition A.1. A matrix $\mathbf{A}$ is Symmetric and Diagonally Dominant (SDD) if $\mathbf{A}_{i,j} = \mathbf{A}_{j,i}$ for all i, j and $\mathbf{A}_{i,i} \geq \sum_{j \neq i} |\mathbf{A}_{\mathbf{i},\mathbf{j}}|$.

The results is a straightforward consequence of Theorem 1.2 from (Orecchia et al., 2012) and Lemma 3.1. For Reader's convenience we restate that theorem here:

Theorem A.2 (SDD Matrix Exponential Computation). *Given an $L \times L$ SDD matrix $\mathbf{A}$, a vector $\mathbf{x}$ and a parameter $\delta \leq 1$, there is an algorithm that computes a vector $\mathbf{u}$ such that $\|\exp(-\mathbf{A})\mathbf{x} - \mathbf{u}\| \leq \delta\|\mathbf{x}\|$ in time $\tilde{O}((|E| + L)\log(2 + \|\mathbf{A}\|))$, Here tilde hides $\mathrm{poly}(\log(L))$ and $\mathrm{poly}(\log(\frac{1}{\delta}))$ factors.*

It suffices to notice that both Laplacian matrix and its renormalized version are SDD. Furthermore, by Lemma 3.1, fast (approximate) computation of $\exp(-\lambda\mathbf{A})\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^L$ and $\mathbf{A}$ as in Theorem 4.1 leads to fast computation of the low-ranked attention with mask $\mathbf{M} = \exp(-\lambda\mathbf{A})$, as explained in Algorithm 1. $\square$

A.10. Proof of Theorem 4.2

Proof. Note first that since $\omega(k)$ and $\omega(l)$ are chosen independently, we have:

$$\mathrm{K}_p^{\lambda,0}(k,l) = \mathbb{E}_{\omega(k)}[f_k^{\omega(k),\lambda}] \cdot (\mathbb{E}_{\omega(l)}[f_l^{\omega(l),\lambda}])^\top = \mathbb{E}_{\omega(k),\omega(l)}[f_k^{\omega(k),\lambda}(f_l^{\omega(l),\lambda})^\top] \tag{17}$$

Denote: $X = f_k^{\omega(k),\lambda}(f_l^{\omega(l),\lambda})^\top$. The key observation is that X can be rewritten as:

$$X = \sum_{u \in \mathrm{V(G)}} \sum_{\substack{(j_1=k,...,j_{a+1}=u)=\mathrm{pref}(\omega(k)),\\ (j'_1=l,...,j'_{b+1}=u)=\mathrm{pref}(\omega(l))}} \lambda^a\lambda^b = \sum_{(j_1=k,...,j_{a+b+1}=l)\in\Omega(k,l)} \lambda^{a+b} \tag{18}$$

where $\Omega(k,l)$ is the multi-set of walks from k to l that are built from some prefix of $\omega(k)$ concatenated with some prefix of $\omega(l)$. Therefore we can write X as:

$$X = \sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \lambda^{\mathrm{len}(r)} 1[\mathcal{E}(r,i)], \tag{19}$$

where $R(k,l)$ is the set of walks from k to l, $\mathrm{len}(r)$ stands for the length (number of edges) of walk r and $\mathcal{E}(r,i)$ is an event that first i edges of the walk r (counting from k) form the prefix sub-walk of $\omega(k)$ and the remaining ones form the prefix sub-walk of $\omega(l)$. Therefore we have:

$$\mathrm{K}_p^{\lambda,0}(k,l) = \mathbb{E}_{\omega(k),\omega(l)}\left[\sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \lambda^{\mathrm{len}(r)} 1[\mathcal{E}(r,i)]\right] =$$
$$\sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \lambda^{\mathrm{len}(r)} \mathbb{P}_{\omega(k),\omega(l)}[\mathcal{E}(r,i)] = \sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \lambda^{\mathrm{len}(r)} \prod_{j=0}^{i-1} \frac{1-p}{\deg(r^j)} \prod_{t=0}^{\mathrm{len}(r)-i-1} \frac{1-p}{\deg(r^{\mathrm{len}(r)-1-t})}, \tag{20}$$

where r^y stands for the y^{th} vertex of the walk r starting from k and $\deg(v)$ denotes the degree of a vertex v.

Therefore we obtain:

$$\sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \left(\frac{(1-p)\lambda}{d_{\max}}\right)^{\mathrm{len}(r)} \leq \mathrm{K}_p^{\lambda,0}(k,l) \leq \sum_{r \in R(k,l)} \sum_{i=0}^{\mathrm{len}(r)} \left(\frac{(1-p)\lambda}{d_{\min}}\right)^{\mathrm{len}(r)} \tag{21}$$

We conclude that:

$$\sum_{i=0}^{\infty} r_{k,l}(i) \left(\frac{(1-p)\lambda}{d_{\max}}\right)^{i} (i+1) \leq \mathrm{K}_p^{\lambda,0}(k,l) \leq \sum_{i=0}^{\infty} r_{k,l}(i) \left(\frac{(1-p)\lambda}{d_{\min}}\right)^{i} (i+1) \tag{22}$$

To complete the proof, it suffices to notice that matrix $\mathrm{Adj}^i(\mathrm{G})$ encodes the number of walks of length i between pairs of vertices in G. $\square$

A.11. Extensions of the results from Section 3.3

We will provide here the proofs of the results presented in Section 6. We first introduce additional concepts wee will leverage in the proofs.

Definition A.3 (balanced separators)**.** Take some function $\mathbf{w} : V(G) \rightarrow \mathbb{R}_{\geq 0}$ and some $\alpha > 0$. We say that a subset $\mathcal{S} \subseteq V(G)$ is the α-balanced separator with respect to $\mathbf{w}$, if the set of vertices $\mathcal{C}$ of every connected component of the subgraph graph $G_{|V(G) \backslash \mathcal{S}}$ of G, induced by $V(G) \backslash \mathcal{S}$, satisfies: $\mathbf{w}(\mathcal{C}) \leq \alpha \cdot \mathbf{w}(V(G))$, where $\mathbf{w}(\mathcal{X}) \overset{\mathrm{def}}{=} \sum_{x \in \mathcal{X}} \mathbf{w}(x)$.

Lemma A.4. *If G is a tree then for an arbitrary function $\mathbf{w} : V(G) \rightarrow \mathbb{R}_{\geq 0}$ the $\frac{1}{2}$-balanced separator consisting of two adjacent vertices can be found in time $O(L)$.*

Proof. The proof is given in the proof of Lemma 7.19 in (Cygan et al., 2015) and relies on the standard tree-search. $\square$

A.11.1. The proof of Lemma 6.1

Proof. Take some vector $\mathbf{x} \in \mathbb{R}^L$. The goal is to compute $\mathbf{M}\mathbf{x}$ in time $O(L\log^2(L))$. For the node i in a tree $\mathcal{T}$, denote:

$$s_i = \sum_{j \in \mathcal{T}} f(\mathrm{dist}(i,j))\mathbf{x}_j \tag{23}$$

Thus we want to compute all s_i in time $O(L\log^2(L))$. If $|V(\mathcal{T})| \leq 2$ then all the calculations can be trivially done in $O(1)$ time, so we will assume now that $|V(\mathcal{T})| > 2$. Take the $\frac{1}{2}$-balanced separator $\{a,b\}$ in $\mathcal{T}$ (with respect to the standard measure that counts the number of vertices) that exists and can be found in time $O(L)$ by Lemma A.4. Denote by T_a the set of those trees in $\mathcal{T}_{|V(\mathcal{T})\backslash\{a,b\}}$ that are are incident to a in $\mathcal{T}$ and by T_b the set of those trees in $\mathcal{T}_{|V(\mathcal{T})\backslash\{a,b\}}$ that are are incident to b in $\mathcal{T}$. Note that one of these sets might be potentially empty. Let us assume, without loss of generality that T_a is not empty. Denote by V_a the union of the set of all the vertices of all the elements of T_a and by V_b the corresponding set for T_b. If $\frac{1}{10} \leq |V_a| \leq \frac{9}{10}|V(\mathcal{T})|$, take: $\mathcal{T}_1$ to be the subtree of $\mathcal{T}$ induced by $V_a \cup \{a\}$ and $\mathcal{T}_2$ to be the subtree of $\mathcal{T}$ induced by $V_b \cup \{a,b\}$. Otherwise take this $c \in \{a,b\}$ such that $|V_c| > \frac{9}{10}|V(\mathcal{T})|$. Denote: $T_c = \{T^1,...,T^m\}$. Note that $m > 0$ (T_c is not empty). By the definition of the balanced separator, we have: $|V(T^i)| \leq \frac{1}{2}|V(\mathcal{T})|$ for $i = 1,...,m$. On the other hand: $|V(T^1)| + ... + |V(T^m)| \geq \frac{9}{10}|V(\mathcal{T})|$. Denote by i^* the smallest $i \in \{1,...,m\}$ such that $|V(T^1)| + ... + |V(T^{i^*})| \geq \frac{9}{10}|V(\mathcal{T})|$. Note that $i^* > 1$. We have:

$$\begin{aligned}\frac{2}{5}|V(\mathcal{T})| = \frac{9}{10}|V(\mathcal{T})| - \frac{1}{2}|V(\mathcal{T})| &\leq |V(T^1)| + ... + |V(T^{i^*})| - |V(T^{i^*})| \\ &= |V(T^1)| + ... + |V(T^{i^*-1})| \leq \frac{9}{10}|V(\mathcal{T})|\end{aligned} \tag{24}$$

Denote by $\mathcal{T}_1$ a subtree of $\mathcal{T}$ induced by $V(T^1) \cup ... \cup V(T^{i^*-1}) \cup \{c\}$ and by $\mathcal{T}_2$ a subtree of $\mathcal{T}$ induced by $V(\mathcal{T}) \backslash (V(T^1) \cup ... \cup V(T^{i^*-1}))$. Note that in both cases we obtain two trees: $\mathcal{T}_1$ and $\mathcal{T}_2$ sharing a single vertex and such that: $V(\mathcal{T}_1) \cup V(\mathcal{T}_2) = V(\mathcal{T})$. Furthermore, we have:

$$|V(\mathcal{T}_1)| = f|V(\mathcal{T})| + c_1, |V(\mathcal{T}_2)| = (1-f)|V(\mathcal{T})| + c_2, \tag{25}$$

for $c_1, c_2 \in \{0,1\}$ and $\frac{2}{5} \leq f \leq \frac{9}{10}$. Denote: $\{v\} = V(\mathcal{T}_1) \cap V(\mathcal{T}_2)$.

Denote: $y_i^1 = \sum_{j \in Z_i^1} \mathbf{x}_j$ and $y_i^2 = \sum_{j \in Z_i^2} \mathbf{x}_j$ for $i = 1, ..., |V(\mathcal{T})|$, where Z_i^k for $k \in \{1,2\}$ stands for the set of vertices in $\mathcal{T}_k$ with distance i from v. Note that all y_i^k can be trivially computed in time $O(|V(\mathcal{T})|)$.

To compute all s_i for $i = 1, ..., |V(\mathcal{T})|$, we first compute recursively the following expressions:

$$s_i^k = \sum_{j \in \mathcal{T}_k} f(\mathrm{dist}(i,j))\mathbf{x}_j \tag{26}$$

for $i \in V(\mathcal{T}_k)$ and $k \in \{1,2\}$. In order to compute expressions s_i, in addition to expressions s_i^k, we need to include the cross-term contributions (for pairs of vertices where one is from $\mathcal{T}_1$ and the other from $\mathcal{T}_2$). Note that this can be trivially done in time $O(V(\mathcal{T}))$ as long as we have computed the following two vectors: $\mathbf{H}\mathbf{y}^1$ and $\mathbf{H}\mathbf{y}^2$, where $\mathbf{y}^k = (y_1^k, ..., y_{|V(\mathcal{T})|}^k)^\top$ for $k \in \{1,2\}$ and $\mathbf{H}$ is the Hankel matrix with the first row of the form: $(2, 3, ..., |V(\mathcal{T})|+1)$ and the last column of the form: $(|V(\mathcal{T})|+1, ..., |V(\mathcal{T})|+|V(\mathcal{T})|)^\top$. This can be done in time $O(|V(\mathcal{T})|\log(|V(\mathcal{T}|)))$ with Fast Fourier Transform. We conclude that our algorithm needs two recursive calls for subproblems of sizes which are constant fractions of $|V(\mathcal{T})|$ and given in Eq. 25, as well as additional computations conducted in time $O(|V(\mathcal{T})|\log(|V(\mathcal{T}|)))$. That leads to the total time complexity $O(|V(\mathcal{T})|\log^2(|V(\mathcal{T}|)))$ which completes the proof. □

A.11.2. THE PROOF OF LEMMA 6.2

Proof. Let us root $\mathcal{T}$ in a fixed vertex v_0. We denote:

$$s_i = \sum_{j \in \mathcal{T}_i} f(\mathrm{dist}(i,j))\mathbf{x}_j, \tag{27}$$

where $\mathcal{T}_i$ stands for the subtree of $\mathcal{T}$ rooted in i. In the first phase of the algorithm, we compute s_i for all $i \in V(\mathcal{T})$. In order to do that, for every node i we maintain an array g_i of length $\mathrm{diam}(\mathcal{T})+1$, where: $g_i[l] = \sum_{j \in \mathcal{T}_i : \mathrm{dist}(i,j)=l} \mathbf{x}_j$. Computing s_i and g_i for vertices i which are the leaves of the tree $\mathcal{T}$ rooted in v_0 can be trivially done in time $O(1)$ per vertex. Now assume that i is not a leaf and denote by: $q_1, ..., q_k$ (for some $k > 0$) its children. Note that: g_i can be computed as follows:

$$g_i[l] = \begin{cases} \sum_{p=1}^k g_{q_p}[l-1], \text{ if } l \geq 1 \\ \mathbf{x}_i, \text{ if } l = 0 \end{cases}$$

Furthermore, s_i can be computes ad follows:

$$s_i = \sum_{l=0}^{\mathrm{diam}(\mathcal{T})} f(l) g_i[l]. \tag{28}$$

We conclude that calculating s_i for all vertices i takes time $O(L \cdot \mathrm{diam}(\mathcal{T}))$. Denote: $\mathbf{w} = \mathbf{M}\mathbf{x}$. Note that $\mathbf{w}_{v_0} = s_{v_0}$. Furthermore, for $i \neq v_0$: $\mathbf{w}_i = s_i + \Gamma_i$, where:

$$\Gamma_i = \sum_{l=1}^{\mathrm{diam}(\mathcal{T})} f(l)(g_{p(i)}[l-1] - g_i[l-2]), \tag{29}$$

where $p(i)$ stands for the parent of i in $\mathcal{T}$ (rooted in v_0) and we take: $g_i[-1] = 0$. We conclude that computing all Γ_i can be done by ordering the vertices of $\mathcal{T}$ by their distance from v_0 and thus can be accomplished in time $O(L \cdot \mathrm{diam}(\mathcal{T}))$. Thus computing all $\mathbf{w}_i$ can be also conducted in time $O(L \cdot \mathrm{diam}(\mathcal{T}))$ and that completes the proof. □