Computationally-efficient Graph Modeling with Refined Graph Random Features

Krzysztof Choromanski [*1 2] **Avinava Dubey** [*3] **Arijit Sehanobish** [*4] **Isaac Reid** [1 5]

Abstract

We propose *refined GRFs* (GRFs++), a new class of *Graph Random Features* (GRFs) for efficient and accurate computations involving kernels defined on the nodes of a graph. GRFs++ resolve some of the long-standing limitations of regular GRFs, including difficulty modeling relationships between more distant nodes. They reduce dependence on sampling long graph random walks via a novel *walk-stitching* technique, concatenating several shorter walks without breaking unbiasedness. By applying these techniques, GRFs++ inherit the approximation quality provided by longer walks but with greater efficiency, trading sequential inefficient sampling of a long walk for parallel computation of short walks and matrix-matrix multiplication. Furthermore, GRFs++ extend the simplistic GRFs walk termination mechanism (Bernoulli schemes with fixed halting probabilities) to a broader class of strategies, applying general distributions on the walks' lengths. This improves approximation accuracy of graph kernels, without incurring extra computational cost. We provide empirical evaluations to showcase our claims and complement our results with theoretical analysis.

1. Introduction & Related Work

Graph modeling plays an important role in several applications of machine learning (ML), such as anomaly, community and fraud detection (Kim et al., 2022; 2026; Beutel et al., 2015; Noble & Cook, 2003; Li et al., 2025; Dong et al., 2025; Chen et al., 2024; Liu et al., 2023), recommender systems (Yang et al., 2025; 2023; Deng et al., 2022; Gao et al., 2023), and computational biology (Mao et al., 2024; Banerjee & Jost, 2009; Zhang et al., 2024). As for Euclidean data, research on graph modeling has spurred the development of many hard-coded/learnable or parameterized (Yanardag & Vishwanathan, 2015) classes of *kernels* (similarity functions), defining relationships between nodes in the graph (Smola & Kondor, 2003; Kondor & Lafferty, 2002) or between the graphs themselves (Vishwanathan et al., 2010; Shervashidze et al., 2009).

In this paper, we focus on graph node kernels $\mathrm{K} : \mathrm{V}(\mathrm{G}) \times \mathrm{V}(\mathrm{G}) \rightarrow \mathrm{R}$ defined on the vertices V of a given graph G, where the similarity between the nodes is measured via their relationship in the graph – e.g how well-connected the nodes are. Common examples of such kernels include the *d-regularized Laplacian*, *diffusion process*, *p-step random walk*, and *inverse cosine* kernels (Smola & Kondor, 2003; Choromanski, 2023). Computing the corresponding Gram matrices $\mathbf{K}(\mathrm{G}) = [\mathrm{K}(v_i, v_j)]_{i,j=1}^N$ for $v_1, ..., v_N \in \mathrm{V}(\mathrm{G})$ tends to be expensive, since this often requires operations of time complexity cubic in the number of graph nodes N. For this reason, research has been dedicated to developing efficient approximation strategies. One common approach is to rewrite the graph kernel as a product of two lower rank matrices, linearizing the graph kernel values with some mapping $\phi : \mathrm{V} \rightarrow \mathbb{R}^m$ as follows:

$$\widehat{\mathrm{K}}(v_i, v_j) = \phi(v_i)^\top \phi(v_j). \tag{1}$$

This low-rank factorization unlocks efficient computations with the corresponding kernel matrices. In particular, matrix-multiplication operations no longer require explicit materialization of $\mathbf{K}$, exploiting the associativity of matrix multiplication. However, until recently this approach was restricted to ad-hoc learnable graph kernels defined implicitly via learnable ϕ (Wu et al., 2019), rather than approximations of the specific classes listed above.

A series of recent papers proposed a new mechanism called *Graph Random Features* (GRFs) (Choromanski, 2023; Reid et al., 2023; 2024b;a; 2025). GRFs provide unbiased approximation of the classes of graph kernels listed above, with probabilistic mappings ϕ obtained via graph random walks. For every graph node v, GRFs build a scalar field on the subset of RW-reachable graph nodes $\mathrm{V}(\mathrm{G})$ via incremental (kernel-dependent) updates of the field in the visited nodes. This field is then mapped to a node-embedding $\phi(v)$, encod-

[*]Equal contribution [1]Google DeepMind, New York, NY, USA [2]Columbia University, New York, NY, USA [3]Google Research, Mountain View, CA, USA [4]Independent Researcher [5]University of Cambridge, Cambridge, UK. Correspondence to: Krzysztof Choromanski <kchoro@google.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

ing the relationship of the node to the entire graph G. Since $\phi(v)$ is probabilistic, it is referred to as the *graph random feature* corresponding to v. Though originally introduced to approximate kernels defined between pairs of nodes, GRFs were recently *lifted* for unbiased approximation of kernels defined between pairs of graphs (Choromanski et al., 2025).

In this paper, we consider a refined class of GRFs, referred to as GRFs++, that addresses several limitations of standard GRFs. The proposed method reduces the reliance on long walks by composing shorter walks, enabling more efficient computation while preserving the core properties of existing GRF methods. We also consider more flexible walk-termination schemes that preserve the unbiasedness of GRFs. The remainder of the paper details these ideas, along with their theoretical and empirical implications.

This paper is organized as follows:

1. In Sec. 2, we present the refined GRFs++ mechanism, introducing the walk-stitching technique (Sec. 2.2.1), a general termination strategy (Sec. 2.2.2), and their connection to higher-order de-convolutions. In Sec. 2 (continued in Sec. 3), we also provide an intrinsic connection between finding a particular instantiation of the GRFs++ algorithm for a given graph kernel and higher-order *(de-) convolutions* of the discrete series encoding its kernel matrix as a Taylor series involving powers of the graph's weight matrices.

2. In Sec. 3, we provide theoretical analysis of GRFs++, including its unbiasedness and concentration results. We show that stitching more walks improves approximation.

3. In Sec. 4, we provide a thorough experimental evidence comparing GRFs++ to regular GRFs on approximation quality, speed, and several downstream tasks: normal vector field prediction on meshes, clustering, graph and image classification.

4. We conclude in Sec. 5 and provide the theoretical proofs in the Appendix (Sec. A).

2. Refined GRFs (GRFs++)

In this section, we start with an overview of GRFs and then systematically derive GRF++, combining several refinements into a unified framework.

2.1. Preliminaries: regular GRFs

We start by providing an overview of the regular GRF mechanism. We take a weighted undirected graph $\mathrm{G}(\mathrm{V}, \mathrm{E}, \mathbf{W} = [w(i,j)]_{i,j \in \mathrm{V}})$ with N nodes/vertices, where (1) V is a set of vertices, (2) $\mathrm{E} \subseteq \mathrm{V} \times \mathrm{V}$ is a set of undirected edges ($(i,j) \in \mathrm{E}$ indicates that there is an edge between i and j in G), and (3) $\mathbf{W} \in \mathbb{R}_{\geq 0}^{N \times N}$ is a weighted adjacency matrix (if no edge exists then the corresponding weight is zero).

We consider the following kernel matrix $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) \in \mathbb{R}^{N \times N}$, where $\boldsymbol{\alpha} = (\alpha_k)_{k=0}^{\infty}$ and $\alpha_k \in \mathbb{R}$:

$$\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) = \sum_{k=0}^{\infty} \alpha_k \mathbf{W}^k. \tag{2}$$

For bounded $(\alpha_k)_{k=0}^{\infty}$ and $\|\mathbf{W}\|_{\infty}$ small enough, the above sum converges. The matrix $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$ defines a kernel on the nodes of the underlying graph. Interestingly, Eq. 2 covers all the special cases of graph node kernels we explicitly listed in Sec. 1. It also covers functions that are not positive definite, since $(\alpha_k)_{k=0}^{\infty}$ can be chosen arbitrarily. From now on, we will associate graph kernels with sequences $(\alpha_k)_{k=0}^{\infty}$.

GRFs enable one to rewrite $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$ (in expectation) as $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) \stackrel{\mathbb{E}}{=} \mathbf{K}_1 \mathbf{K}_2^{\top}$, for independently sampled $\mathbf{K}_1, \mathbf{K}_2 \in \mathbb{R}^{N \times d}$ and some $d \leq N$. This factorization enables efficient (sub-quadratic) and unbiased approximation of the matrix-vector products $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\mathbf{x}$ as $\mathbf{K}_1(\mathbf{K}_2^{\top}\mathbf{x})$, if $\mathbf{K}_1, \mathbf{K}_2$ are sparse or $d = o(N)$. This is often the case in practice. However, if this does not hold, explicitly materializing $\mathbf{K}_1 \mathbf{K}_2^{\top}$ enables one to approximate $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$ in quadratic (c.f. cubic) time. Below, we describe the base GRF method for constructing sparse $\mathbf{K}_1, \mathbf{K}_2$ for $d = N$. Extensions giving $d = o(N)$, using the Johnson-Lindenstrauss Transform (Freksen, 2021), can be found in (Choromanski, 2023). Each $\mathbf{K}_j$ for $j \in \{1, 2\}$ is obtained by row-wise stacking of the vectors $\phi_f(i) \in \mathbb{R}^N$ for $i \in \mathrm{V}$, where f is the *modulation function* $f : \mathbb{R} \to \mathbb{R}$, specific to the graph kernel being approximated. The procedure to construct random vectors $\phi_f(i)$ is given in Algorithm 1. Intuitively, one samples an ensemble of RWs from each node $i \in V$. Every time a RW visits a node, the scalar value in that node is updated with the fraction of the load carried by the walker, depending on the modulation function.

After all the walks terminate, the vector $\phi_f(i)$ is obtained by concatenation of all the scalars/loads from the discrete scalar field, followed by a simple renormalization. It remains to describe how the kernel-dependent modulation function f is constructed. For unbiased estimation, $f : \mathbb{N} \to \mathbb{C}$ needs to satisfy $\sum_{p=0}^{k} f(k-p)f(p) = \alpha_k$, for $k = 0, 1, ...$ (see Theorem 2.1 in (Reid et al., 2024b)).

2.2. From GRFs to GRFs++

We now introduce GRF++ through a walk stitching mechanism (Sec. 2.2.1), extend the underlying Bernoulli trial scheme (Sec. 2.2.2), and finally combine these components into a unified construction 2.2.3.

Algorithm 1 **Regular GRFs:** Construct vectors $\xi_\rho(i) \in \mathbb{R}^N$ to approximate $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$

Input: weighted adjacency matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$, vector of unweighted node degrees (number of out-neighbours) $\deg \in \mathbb{R}^N$, modulation function $\rho : (\mathbb{N} \cup \{0\}) \rightarrow \mathbb{R}$, termination probability $p_{\text{halt}} \in (0,1)$, node $i \in \mathcal{N}$, number of random walks to sample $m \in \mathbb{N}$.
Output: signature vector $\xi_\rho(i) \in \mathbb{R}^N$.

1: initialize: $\xi_\rho(i) \leftarrow \mathbf{0}$
2: **for** $w = 1, ..., m$ **do**
3: initialize: load $\leftarrow 1$, current_node $\leftarrow i$, terminated $\leftarrow$ False, walk_length $\leftarrow 0$
4: **while** terminated $=$ False **do**
5: $\xi_\rho(i)[\text{current_node}] \leftarrow \xi_\rho(i)[\texttt{current_node}] + \texttt{load} \times \rho(\text{walk_length})$
6: walk_length $\leftarrow \texttt{walk_length} + 1$
7: new_node $\leftarrow$ Unif $[\mathcal{N}(\text{current_node})]$ ▷ assign to one of neighbors
8: load $\leftarrow \texttt{load} \times \frac{\deg[\text{current_node}]}{1-p_{\text{halt}}} \times \mathbf{W}[\text{current_node}, \text{new_node}]$ ▷ update load
9: current_node $\leftarrow$ new_node
10: terminated $\leftarrow (t \sim \text{Unif}(0,1) < p_{\text{halt}})$ ▷ draw RV t to decide on termination
11: **end while**
12: **end for**
13: normalize: $\xi_\rho(i) \leftarrow \xi_\rho(i)/m$

2.2.1. Walk-stitching mechanism

The inherently sequential procedure of constructing random walks is not supported by modern accelerators. This is one of the key weaknesses of regular GRFs. Shortening the walks by increasing p_{halt} can in principle mitigate this, at the cost of giving up modeling relationships between more distant nodes in the graph; a graph kernel value between two nodes i and j whose corresponding walks do not intersect is approximated by zero.

In GRFs++, we propose a novel *walk-stitching* technique, where several independently-calculated shorter walks are combined to emulate sampling a longer walk. Mathematically, we unbiasedly approximate graph kernel matrix $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$ as:

$$\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) \stackrel{\mathbb{E}}{=} \prod_{i=1}^{l} \mathbf{K}_1^{(i)} (\mathbf{K}_2^{(i)})^\top, \tag{3}$$

We refer to $l \in \mathbb{N}_+$ as the walk-stitching *degree*. Each i corresponds to one pair of intersecting walks from the regular GRF mechanism. GRFs++ with degree $l = 1$ are equivalent to regular GRFs. A schematic is given in Fig. 1.

Each $\mathbf{K}_j^{(i)}$, for $j \in \{1, 2\}$, is computed as described in Algorithm 1, but the modulation function changes. The following is true:

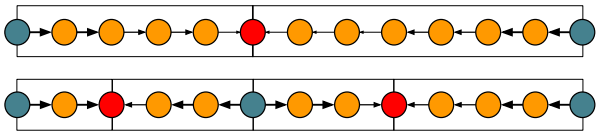

Figure 1. Pictorial description of the *walk-stitching technique*. Each rectangular block corresponds to a random walk and red nodes depict vertices where walks meet. The blue nodes are the communicating ones. The thickness of the arrow, depicting a transition from step t to step $t+1$, indicates the probability that such a transition will occur (a walk can terminate earlier). **Top:** In regular GRFs, two graph vertices communicate via intersecting walks, originating at each vertex. As the nodes become more distant, the probability that such two walks will be constructed decreases. **Bottom:** In GRFs++, two nodes communicate with each other less directly, via proxies (the middle blue node in the picture) and much shorter walks, with lengths that have much higher probability of being realized. The communication is established by stitching several small walks.

Lemma 2.1 (Unbiased walk-stitching and higher-level convolutions)**.** *Assume for each independent instantiation of Alg. 1, the modulation function f satisfies:*

$$\alpha_k = \sum_{p_1 + p_2 + \ldots + p_{2l} = k} f(p_1) f(p_2) \ldots f(p_{2l}). \tag{4}$$

Then the product $\prod_{i=1}^{l} \mathbf{K}_1^{(i)} (\mathbf{K}_2^{(i)})^\top$ provides an unbiased estimation of $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$.

We prove Lemma 2.1 (in fact its more general version) in Sec. 3. The condition from Lemma 2.1 is equivalent to saying that coefficients α_k are obtained via $2l$-level discrete convolution $\overbrace{(f \star f) \ldots (f \star f)}^{l}$ of the modulation function f with itself. Equivalently, the function f must be constructed by $2l$-*de-convolving* sequence $\boldsymbol{\alpha} = (\alpha_k)_{k=0}^{\infty}$ that defines the graph kernel.

Interestingly, for several classes of graph kernels this deconvolution can be efficiently calculated. For example, for GRFs++ with an l-degree walk-stitching mechanism applied to graph diffusion kernels of the form $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) = \exp(\lambda \mathbf{W})$, the modulation function admits a simple closed form: $f(p) = \frac{\lambda^p}{(2l)^p p!}$. In Sec. 3, we provide a general mechanism to find f for more arbitrary $\mathbf{K}_{\boldsymbol{\alpha}}$.

2.2.2. Going beyond the Bernoulli Trial Scheme

Another key building block of GRFs is the *walk termination mechanism*. In regular GRFs, walk lengths are built incrementally, with walkers terminating independently with probability p_{halt} at each timestep. This gives the update rule in line 10 of Algorithm 1. However, sampling walk lengths from the Bernoulli distribution is not necessarily optimal

given fixed computational budget (e.g. fixed average walk length). Here, we propose a general scheme of RW-length sampling, via a simple modification to the update step in Algorithm 1 that improves kernel estimation accuracy.

Take any discrete probabilistic distribution on $\mathbb{N}$: $\mathbf{P} = (P(i))_{i=0}^{\infty}$. We will only assume that: (1) sampling $X \sim \mathbf{P}$ and (2) the computation of $\mathbb{P}(X \geq k)$ for any given $k \in \mathbb{N}$ can be conducted efficiently. We modify Algorithm 1 as follows, to obtain Algorithm 2:

1. The m lengths of walks are sampled: $s_1, ..., s_m \overset{\text{iid}}{\sim} \mathbf{P}$ before line 2.
2. Line 5 is updated as follows, for $\tau(k) \stackrel{\text{def}}{=} \mathbb{P}(X \geq k)$: $\phi_f(i)[\text{current_node}] \leftarrow \phi_f(i)[\text{current_node}] + \frac{\text{load} \times f(\text{walk_length})}{\tau(\text{walk_length})}$.
3. In line 8, term $1 - p_{\text{halt}}$ is dropped from the update equation.
4. Line 10 is updated as follows: terminated $\leftarrow \mathbb{I}[\text{walk_length} \geq s_m]$.

Note that Algorithm 1 is a special instantiation of Algorithm 2, with $\mathbf{P}$ corresponding to the number of consecutive successes of a Bernoulli scheme with failure probability p_{halt}. In Sec. 3, we show that Lemma 2.1 still holds if Algorithm 1 is replaced by Algorithm 2.

2.2.3. Putting it all together

We now present the complete GRFs++ mechanism, which we will refer to as $(l, \mathbf{P})$-GRFs++, where $l \in \mathbb{N}_+$ and $\mathbf{P} \in \mathcal{P}(\mathbb{N})$ are the hyperparameters of the mechanism. We first construct $(\mathbf{K}1^i, \mathbf{K}2^i)_{i=1}^{l}$ as in Lemma 2.1, with Algorithm 2 replacing Algorithm 1.

Option I: In the most direct approach, the refined random feature vectors are given as rows of the following two matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N \times N}$, satisfying $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) \stackrel{\mathbb{E}}{=} \mathbf{X}\mathbf{Y}^\top$:

$$\mathbf{X} = \prod_{i=1}^{\frac{l}{2}} \mathbf{K}_1^{(i)} (\mathbf{K}_2^{(i)})^\top, \mathbf{Y} = \prod_{i=l}^{\frac{l}{2}+1} \mathbf{K}_2^{(i)} (\mathbf{K}_1^{(i)})^\top, \text{ if } l \text{ is even}$$

$$\mathbf{X} = \left[\prod_{i=1}^{\frac{l-1}{2}} \mathbf{K}_1^{(i)} (\mathbf{K}_2^{(i)})^\top \right] \mathbf{K}_1^{(\frac{l+1}{2})}, \tag{5}$$

$$\mathbf{Y} = \left[\prod_{i=l}^{\frac{l+3}{2}} \mathbf{K}_2^{(i)} (\mathbf{K}_1^{(i)})^\top \right] \mathbf{K}_2^{(\frac{l+1}{2})}, \text{ if } l \text{ is odd.} \tag{6}$$

We define the product of the empty sequence of matrices as an identity matrix. If all the matrices $\mathbf{K}_j^{(i)}$ are sparse (i.e. contain only linear in N number of nonzero entries; note for instance that for the regular p_{halt}-termination strategy, the average number of those entries is $Nm\frac{1-p_{\text{halt}}}{p_{\text{halt}}}$), then $\mathbf{X}, \mathbf{Y}$ can be computed in time $O(N^2)$ for constant l and are also sparse. This means the refined random feature vectors are sparse, like their regular counterparts.

Option II: Like regular GRFs (Choromanski, 2023), the Johnson-Lindenstrauss Transform (JLT)(Freksen, 2021) can be used to reduce the dimensionality of GRFs++, at the cost of sacrificing their sparsity. The formula for matrices $\mathbf{X}, \mathbf{Y}$ is analogous to this from Option I, but with matrices $\mathbf{K}_j^{(i)}$ replaced by their down-projections, obtained with random Gaussian variates. In particular, we take

$$\widehat{\mathbf{K}}_j^{(i)} = \frac{1}{\sqrt{r}} \mathbf{K}_j^{(i)} \mathbf{G}^{(i)}, \tag{7}$$

for independently created Gaussian matrices $\mathbf{G}^{(i)} \in \mathbb{R}^{N \times r}$, with entries taken independently at random from $\mathcal{N}(0,1)$ and a hyperparameter $r \in \mathbb{N}$. For constant l, r, the computation of all $\widehat{\mathbf{K}}_j^{(i)}$ can be done in $O(N^2)$ time (with no sparsity assumption on $\mathbf{K}_j^{(i)}$). Note also, that under this condition, matrices $\mathbf{X}, \mathbf{Y}$ can be computed in time $O(N)$, via matrix associativity property. Since the JLT preserves dot-products in expectation, we conclude that $\mathbb{E}[\widehat{\mathbf{K}}_1^{(i)} \widehat{\mathbf{K}}_2^{(i)}] = \mathbb{E}[\mathbf{K}_1^{(i)} \mathbf{K}_2^{(i)}]$ for each i. Thus resulting $\mathbf{X}, \mathbf{Y}$ still satisfy: $\mathbf{K}_\alpha(\mathbf{W}) = \mathbb{E}[\mathbf{X}\mathbf{Y}^\top]$.

Option III: In practice, as for regular GRFs, GRFs++ do not always need to be explicitly constructed. In most applications of random feature methods, one only needs access to products between the (approximate) kernel matrix and vectors, rather than the kernel matrix itself. As such, one only needs to support efficient multiplication algorithm for $\left[\prod_{i=1}^{l} \mathbf{K}_1^{(i)} \mathbf{K}_2^{(i)}\right] \mathbf{v}$ for any $\mathbf{v} \in \mathbb{R}^N$. This can be done by multiplying with matrices from the chain $\prod_{i=1}^{l} \mathbf{K}_1^{(i)} \mathbf{K}_2^{(i)}$ or the chain $\prod_{i=1}^{l} \widehat{\mathbf{K}}_1^{(i)} \widehat{\mathbf{K}}_2^{(i)}$ from right to left, exploiting associativity. If we use the setting from Option I, l is constant and individual matrices are sparse, so this can be done in time $O(N)$ (rather than brute-force $O(N^2)$). This is also the case if Option II is applied with constant r.

Parallel computations of random walks in GRFs++: One of the most attractive computational features of GRFs++ is that one can compute short RWs in parallel for different $i = 1, 2, ..., l$. These are in turn put together by walk-stitching, implicitly constructing longer walks. This gives computational gains compared to regular GRFs, since it avoids explicit, sequential sampling of longer walks.

Re-using the same set of random walks: Although unbiasedness requires the matrices $\mathbf{K}_j^{(i)}$ for $j \in 1, 2$ to be constructed from independent sets of random walks, we empirically find that re-using the same set of random walks

performs well in practice. This effect is particularly pronounced for larger graphs with higher diameter (Sec. 4).

Walk-stitching with general termination strategies: To see how walk-stitching helps more distant nodes to connect with each other, consider a termination strategy, where the first transition occurs with probability $p_0 = 1$ and consequent transitions occur with probability $p_{\text{next}} < 1$. In such a setting, the probability of regular GRFs emulating any existing walk of length $r \geq 2$, joining two given vertices i and j scales with p_{next} as p_{next}^{r-2}, whereas for walk-stitching of degree $\lceil \frac{r}{2} \rceil$, this walk will be emulated via GRFs++ with probability lower-bounded by the expression completely independent of p_{next}.

3. Theoretical analysis

We now provide a rigorous theoretical analysis of GRFs++. We start by presenting a strengthened version of Lemma 2.1 from Section 2 (proof in App. A.1).

Lemma 3.1 (Unbiased walk-stitching, higher-order convolutions & general termination)**.** *Lemma 2.1 remains true if Algorithm 1 in its statement is replaced by Algorithm 2.*

Since Algorithm 2 is more general than Algorithm 1 (Sec. 2.2.2), this also proves Lemma 2.1. The setting with Algorithm 1, $l = 1$ is equivalent to regular GRFs.

The formula for the mean squared error (MSE) of the GRFs++-based graph kernel estimator with general degree $l \geq 1$ in terms of the individual components $\mathbf{X}_i = \mathbf{K}_1^{(i)} \mathbf{K}_2^{(i)}$ is complicated. However, for degree $l = 2$, it has a compact form, provided below.

Lemma 3.2 (MSE of the approximation via GRFs++ with $l = 2$)**.** *The* MSE *of the estimator* $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})$ *of the groundtruth graph kernel matrix* $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$*, leveraging GRFs++ with degree* $l = 2$ *satisfies (proof in the Appendix: Sec. A.2,* $\|\|_{\mathrm{F}}$ *stands for the Frobenius norm):*

$$\begin{aligned}\mathrm{MSE}(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})) &\stackrel{\text{def}}{=} \mathbb{E}[\|\mathbf{X}_1\mathbf{X}_2 - \mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2] \\ &= \|\mathbb{E}[\mathbf{X}_1^\top \mathbf{X}_1]\|_{\mathrm{F}}^2 - \|\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2 \quad (8)\end{aligned}$$

Finally, we show that the approximation of the graph kernel improves with GRFs++ degree (for degrees being the powers of two; proof in App. A.3).

Theorem 3.3. *If* $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(l)}(\mathbf{W})$ *stands for the estimator of the ground-truth graph kernel matrix, leveraging GRFs++ of degree* l*, then the following holds if standard termination strategy is applied:*

$$\begin{aligned}\mathrm{MSE}(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(1)}(\mathbf{W})) &\geq \mathrm{MSE}(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2)}(\mathbf{W})) \\ &\geq \mathrm{MSE}(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(4)}(\mathbf{W})) \geq \ldots \quad (9)\end{aligned}$$

3.1. De-mystifying $2l$-level de-convolutions

Let us assume that the coefficient $\boldsymbol{\alpha} = (\alpha_k)_{k=0}^{\infty}$ defining graph kernel, encode also an analytical function $g : \mathbb{C} \to \mathbb{C}$ of the form: $g(x) = \sum_{i=0}^{\infty} \alpha_k x^k$. Assume furthermore that one can compute $h(x) = g^{\frac{1}{2l}}(x)$ and its Taylor expansion is of the form: $h(x) = \sum_{i=0}^{\infty} \beta_i x^i$. Then it is easy to see that function: $f(p) = \beta_p$ satisfies Equation 4.

The above observation provides a straightforward algorithm for computing modulation function f for GRFs++ with hyperparameter l: (1) map the graph kernel under consideration to function g, (2) compute its $(2l)^{th}$-root h, (3) find Taylor series of h to define f.

Remark 3.4. We now see why the expression for f in the GRFs++ mechanism associated with the diffusion graph kernel is particularly simple for any $l \in \mathbb{N}_+$. This follows from the fact that the l-th root of the generating function $g(x) = \exp(x)$ has the closed form $g^{1/l}(x) = \exp(x/l)$.

4. Experiments

In this section, we showcase the ability of GRFs++ to efficiently approximate graph node kernels (see Sec. 4.1), including with larger diameters. Furthermore, we show downstream applications of GRFs++ in graph classification, node clustering tasks and normal field prediction on meshes (see Sec 4.2). We use the graph diffusion kernel for all experiments in this section. Furthermore, we provide kernel estimation for other kernels including : (1) $(\mathbf{I} - \mathbf{W})^{-1}$, (2) $(\mathbf{I} - \mathbf{W}^2)^{-1}$, (3) $\exp(\frac{\mathbf{W}^2}{4})$ in Sec. B.6 (Fig. 9, 10, and 11). For all these kernels, GRF++ outperform GRF.

4.1. Accurate estimation of Graph Kernels with GRFs++

Following (Reid et al., 2024b), we choose eight graphs of varying sizes: (1) Erdős-Rényi graphs of two sizes, (2) a binary tree, (3) a d-regular graph, and (4) four real world examples (karate, dolphins, football and eurosis). Fig. 2 plots the relative Frobenius norm error of the approximation $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})$ of the groundtruth kernel matrix $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$ with GRFs++ (i.e., $\|\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W}) - \widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}} / \|\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}$) against the number of random walks m, showcasing improved estimation accuracy with GRFs++. Next, we show that our method can capture long-distance information more accurately than regular GRFs (Fig. 7). For this task, we select eight diverse graphs from datasets including Peptides (Dwivedi et al., 2022), CIFAR-10 (Dwivedi et al., 2023), Reddit-Binary (Morris et al., 2020), and Geometric Shapes (Yannick-S, 2025). These graphs exhibit varying degrees of sparsity and heterophily, yet all are characterized by large diameters (up to 159). We again compute kernel estimation error (with $p_{\text{halt}} = 0.1$), but now for node pairs that are a specified distance apart. Again, GRFs++ are more

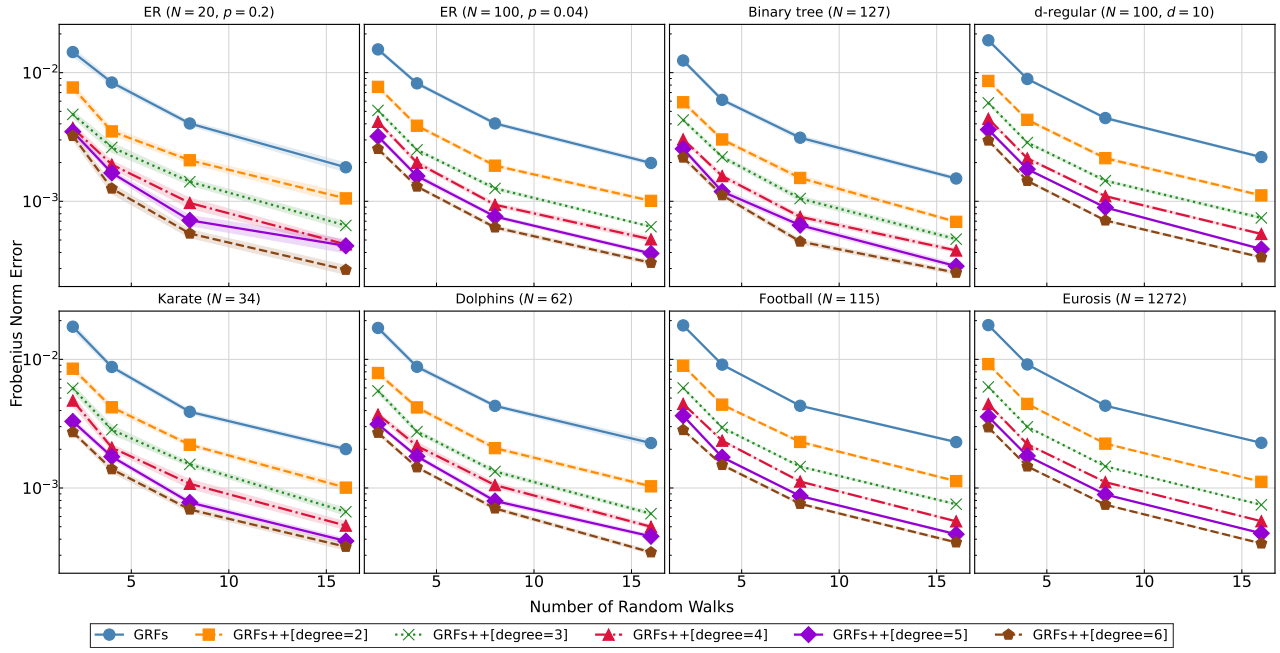

Figure 2. Comparison of different GRF methods for the diffusion kernel estimation. The approximation error (y-axis) improves with the number of walks m (x-axis) and GRF++ provides a sharper estimate than the previous GRF mechanism. Experiments repeated 10 times.

accurate. See App. B.1.1 for details.

New Termination Strategy: Next, we investigate the benefits a more general (non-Bernoulli) termination strategy, as described in Sec. 2.2.2. Specifically, we employ a halting probability governed by a **Poisson distribution** $\mathbf{P}$. For a fair comparison, the parameters are chosen so that the expected random walk length remains the same as in regular GRFs. Fig. 3 shows that our novel halting strategy improves regular GRF mechanism on a wide range of diverse graphs. We also get a more accurate estimation of kernel values for distant nodes in large diameter graphs (see Fig. 8).

Re-using the same set of random walks : Finally, we conduct an ablation study (Fig. 5 **(a)**) to pinpoint the benefit of the walk-stitching mechanism itself. In this experiment, rather than sampling new independent walks for each component (as before), we use **the exact same set** of random walks generated for the baseline GRF method and re-use them for each degree of the GRFs++ estimator. This results in a significant improvement in the Frobenius norm error for all GRFs++ variants, compared to the regular GRFs baseline (see Tab. 10 for a practical downstream experiment).

Computational Time : We generated random graphs with 500 nodes. We evaluated two configurations using base halting probabilities (p_{halt}) of 0.01 and 0.001, which correspond to the regular GRF method (degree $l = 1$). For the GRFs++ methods of degree $l > 1$, we used a scaled halting probability of $p_{\text{halt}} \times l$ to ensure a fair comparison. Fig. 5 **(b)** presents a computational speed analysis of the GRFs++ algorithm, breaking down its performance into two key components: "Walk Time" (the time required for random walk sampling) and "Stitching Time" (the time for the matrix-matrix operations used in the walk-stitching technique). As the degree increases, both the walk time and the stitching time decreases.

Table 1. **Runtime comparison of brute-force (BF) computation and GRF++ on meshes of increasing size.** GRF++ consistently achieves substantial speedups and remains tractable even when the brute-force baseline runs out of time (OOT). **OOT**: execution exceeded the allotted runtime budget.

Mesh Size	BF Runtime (s)	GRF++ Runtime (s)
11,700	1,661.74	**314.98**
15,527	4,833.51	**727.48**
24,194	14,062.08	**1,724.71**
35,300	OOT	**5,591.42**

We further compare the speed of GRF++ against the brute force kernel construction. Tab. 1 shows scalabilty of GRF++ for larger graphs (results for smaller meshes can be found in Fig. 6).

4.2. Downstream Tasks

We show the efficacy of our GRFs++ based approximate kernel in various downstream tasks: image classification, graph

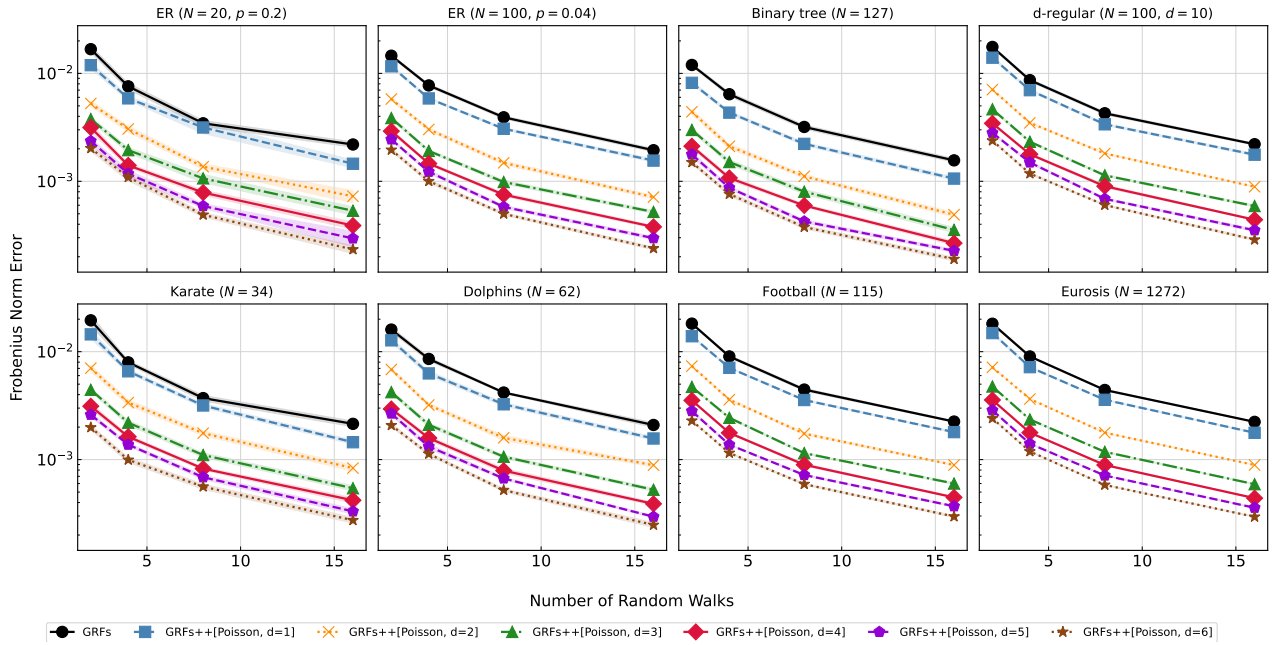

Figure 3. Our novel halting policy based on Poisson distribution provides additional gains over the GRF mechanisms. We run the experiment $s = 10$ times on different graphs of varying sizes.

classification, node clustering and vertex normal prediction.

Image Classification : In this experiment, we investigate the utility of GRFs++ for introducing inductive biases into the self-attention mechanism of Vision Transformers (ViTs) (Dosovitskiy et al., 2021). We treat the input image patches as nodes in a regular 2D grid graph (lattice), where edges connect spatially adjacent patches. We employ GRFs++ (with degree $l = 2$) to efficiently approximate the diffusion kernel matrix $\mathbf{K}_{\text{diff}}$ on this grid graph. This approximate kernel is subsequently normalized and utilized as a soft mask $\mathbf{M}$, which is added to the standard attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \lambda \mathbf{M}\right)\mathbf{V}$$

We consistently get gains over the baseline ViT on ImageNet (Deng et al., 2009) and Places365 (Zhou et al., 2017) (Tab. 3) with additional details in App. B.5.

Graph Classification : Graph kernels have been widely used for graph classification tasks (Kriege et al., 2020; Nikolentzos et al., 2021). We compare the graph classification results obtained using the approximate kernel from GRF++ with those from the exact diffusion kernel on a wide variety of datasets (Morris et al., 2020). Fig. 4 shows that our method performs on par with the diffusion kernel and outperforms regular GRF (additional details in App. B.2). In Tables 8 and 9, we report the results for a wide range of baselines. We observe that GRF++ achieves competitive performance among various strong kernel-based classification baseline approaches. Note that GRF++ results are not directly comparable with other approaches, as GRF++ constructs an intra-graph kernel while other methods use inter-graph kernels. Despite the aforementioned considerations, we contend that positioning our results within the broader framework of alternative methodologies demonstrates that GRF++ remains a compelling approach, owing to its speed and comparable classification accuracy.

Node Clustering : We also test GRFs++'s utility on the downstream task of node clustering. For this experiment, we perform spectral clustering, implemented using the SciPy library, to group nodes based on the approximated diffusion kernel. We compare the clustering error $E = (\#\ of\ wrong\ pairs)/(N * (N-1))$ of the baseline GRF with our GRF++[degree=2] method. The results, presented in Tab. 4 , show that GRF++ achieves a lower error rate **across all tested datasets** (additional details in App. B.3). Furthermore, we compare our method with additional clustering methods like LOUVAIN, SPECTRAL, KCENTERS and PROPAGATION in Table 11.

Normal Prediction : We test GRF++ on normal vector prediction (mesh interpolation). Every vertex of the graph mesh G with a vertex-set V, is associated with spatial coordinates $\mathbf{x_i} \in \mathbb{R}^3$ and a unit normal vector $\mathbf{F}_i \in \mathbb{R}^3$. Following (Choromanski et al., 2024), we randomly sample a subset $V' \subset V$ from each mesh with $|V'| = 0.8|V|$ and mask out their vertex normals. Our goal is to pre-

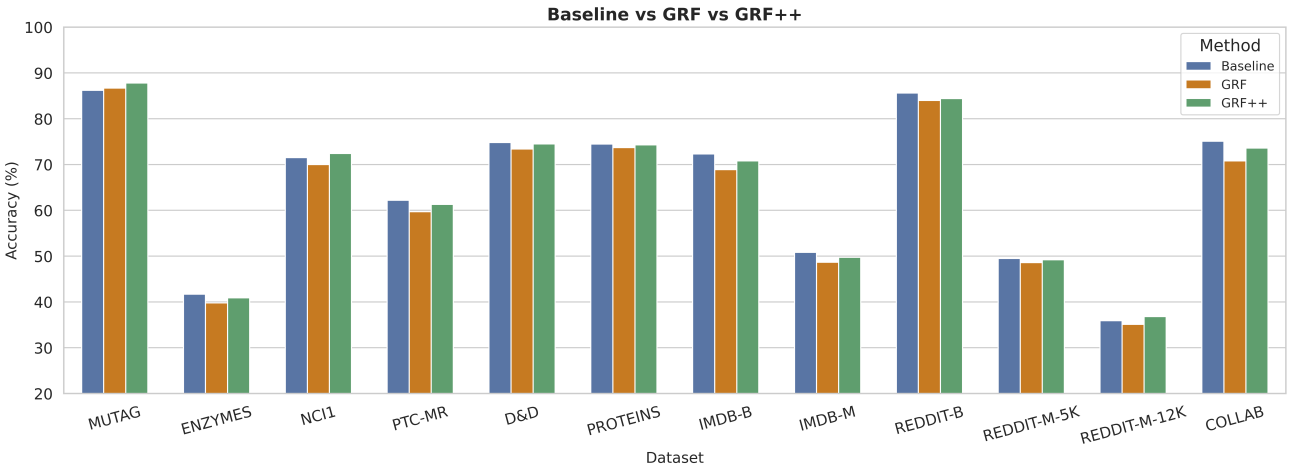

Figure 4. Graph classification using the approximate diffusion kernel from GRF++. Our method performs at par with the baseline diffusion kernel and always beats GRFs on 12 different graph datasets.

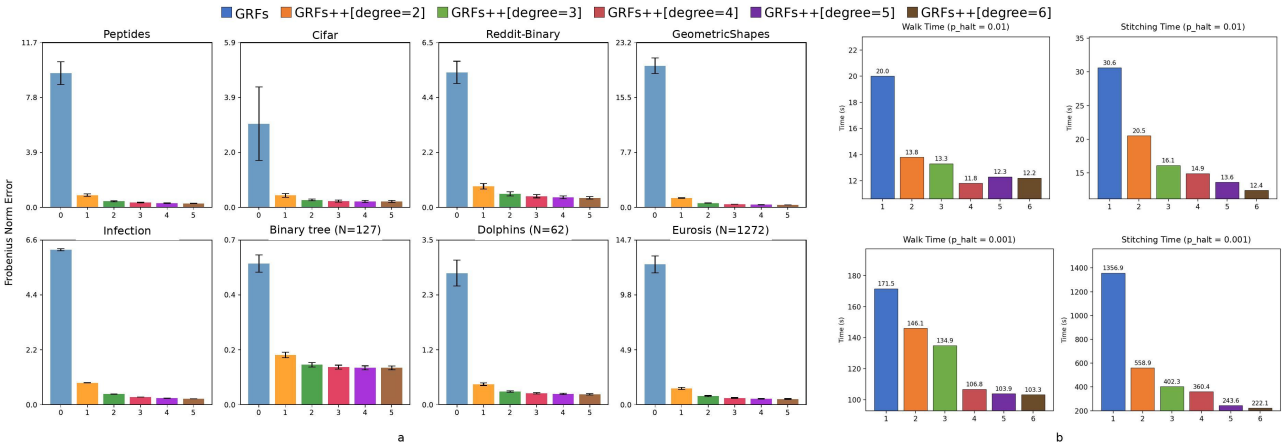

Figure 5. a) Using the exact same walk as the baseline GRF, repeated multiple times, pinpoints the effectiveness of the walk-stitching algorithm, showing additional computational gains. b) Speed comparison for various GRF-methods: regular GRFs and GRFs++ with different degrees.

Table 2. Cosine Similarity results for Meshes. GRF++ matches the baseline kernel (BF) and outperforms GRFs. GRF++r, reusing the same random walk, also outperforms GRF.

MESH SIZE	5985	6577	6911	7386	7953	8011	8261	8449	8800	9603
BF	0.9194	0.9622	0.9769	0.9437	0.9460	0.9382	0.9196	0.9276	0.9836	0.9766
GRF	0.9091	0.9525	0.9682	0.9308	0.9383	0.9233	0.9050	0.9139	0.9778	0.9708
GRF++	0.9154	0.9599	0.9751	0.9374	0.9429	0.9321	0.9145	0.9205	0.9820	0.9748
GRF++r	0.9129	0.9561	0.9701	0.9348	0.9410	0.9269	0.9095	0.9160	0.9805	0.9734
Diff	0.0063	0.0074	0.0069	0.0066	0.0046	0.0088	0.0095	0.0066	0.0042	0.0040

dict the vertex normals of each masked vertex $i \in V'$ via: $F_i = \sum_{j \in V \setminus V'} \mathrm{K}(i,j) F_j$, where K is the diffusion kernel. We report the cosine similarity between predicted and groundtruth vertex normals, averaged over all the nodes. We validate GRF++ over **40** meshes of 3D printed objects of varying sizes from the Thingi10K dataset (Zhou & Jacobson, 2016). Additional details are provided in App. B.4. To save space, we show all the results in Tab. 10 in App B.4 presenting a few larger meshes in Table 2. GRFs++ provide consistent gains, compared to regular GRFs. Moreover we show that GRF++ provides better scaling than the baseline kernel (Fig. 6).

GRF++ in Graph Transformers : To further validate our approach, we evaluate GRF++ within the GraphGPS framework (Rampášek et al., 2022), using a Performer back-

Table 3. Top-1 accuracy on ImageNet and Places365 validation sets. We compare the standard Vision Transformer (ViT) against our GRF++ variant. GRF++ consistently outperforms the baseline across both datasets.

Dataset	ViT	GRF++
ImageNet	80.10	**80.31**
Places365	55.78	**56.03**

Table 4. Node Clustering: GRFs++ vs regular GRFs. GRF++ outperforms GRF on all the datasets.

Name	# Nodes	# clusters	GRF	GRF++[d=2]
Karate	34	2	0.2995	**0.2585**
Dolphins	62	2	0.0635	**0.0323**
Polbooks	105	3	0.1060	**0.1033**
Football	115	12	0.0731	**0.0362**
Databases	1006	6	0.3528	**0.3001**
Eurosis	1272	13	0.2248	**0.1304**

bone without any default positional encoding. As shown in Tab. 5, integrating GRF++ consistently improves upon this baseline across established benchmarks (Dwivedi et al., 2022; 2023). These results confirm that GRF++ generates highly effective positional node representations, demonstrating that the utility of our method extends well beyond kernel approximation. Additional details are presented in App. C.

5. Conclusion

We introduced *refined GRFs* (GRFs++) for improved approximation of graph kernels. GRFs++ address regular GRFs' shortcomings, modeling the relationship between distant pairs of nodes more effectively and introducing novel random walk termination strategies. GRFs++ provide more accurate and more efficient kernel approximation, replacing computationally-inefficient and inherently sequential sampling of long random walks with matrix-matrix operations. We complement our algorithm with theoretical analysis, showing that GRFs++ give unbiased approximation. We provide concentration results, as well as detailed empirical evaluation on a wide variety of graph datasets and tasks.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Banerjee, A. and Jost, J. Graph spectra as a systematic tool in computational biology. *Discret. Appl. Math.*, 157 (10):2425–2431, 2009. doi: 10.1016/J.DAM.2008.06.033. URL https://doi.org/10.1016/j.dam.2008.06.033.

Beutel, A., Akoglu, L., and Faloutsos, C. Graph-based user behavior modeling: From prediction to fraud detection. In Cao, L., Zhang, C., Joachims, T., Webb, G. I., Margineantu, D. D., and Williams, G. (eds.), *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pp. 2309–2310. ACM, 2015. doi: 10.1145/2783258.2789985. URL https://doi.org/10.1145/2783258.2789985.

Chen, J., Zhu, G., Yuan, C., and Huang, Y. Boosting graph anomaly detection with adaptive message passing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=CanomFZssu.

Choromanski, K. M. Taming graph kernels with random features. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 5964–5977. PMLR, 2023. URL https://proceedings.mlr.press/v202/choromanski23a.html.

Choromanski, K. M., Sehanobish, A., Chowdhury, S. B. R., Lin, H., Dubey, K. A., Sarlos, T., and Chaturvedi, S. Fast tree-field integrators: From low displacement rank to topological transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Eok6HbcSRI.

Choromanski, K. M., Reid, I., Sehanobish, A., and Dubey, K. A. Optimal time complexity algorithms for computing general random walk graph kernels on sparse graphs. In Li, Y., Mandt, S., Agrawal, S., and Khan, M. E. (eds.), *International Conference on Artificial Intelligence and Statistics, AISTATS 2025, Mai Khao, Thailand, 3-5 May 2025*, volume 258 of *Proceedings of*

Table 5. Performance comparison between Performer and our GRF-based variants across benchmark datasets. Lower is better for Peptides-func.

Dataset	**Performer**	**+ GRF**	**+ GRF++**
CIFAR-10	69.96	70.10	**70.21**
PATTERN	84.91	85.20	**85.39**
CLUSTER	75.75	76.08	**76.28**
MNIST	97.02	97.14	**97.38**
Peptides-func ↓	27.84	26.88	**26.61**
Peptides-struct	62.93	64.17	**64.33**

Machine Learning Research, pp. 3457–3465. PMLR, 2025. URL https://proceedings.mlr.press/v258/choromanski25a.html.

de Lara, N. and Pineau, E. A simple baseline algorithm for graph classification, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Deng, L., Lian, D., Wu, C., and Chen, E. Graph convolution network based recommender systems: Learning guarantee and item mixture powered strategy. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Dong, X., Zhang, X., Chen, L., Yuan, M., and Wang, S. Spacegnn: Multi-space graph neural network for node anomaly detection with extremely limited labels. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=Syt4fWwVm1.

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 *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=in7XC5RcjEn.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24 (43):1–48, 2023.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

Freksen, C. B. An introduction to johnson-lindenstrauss transforms. *CoRR*, abs/2103.00564, 2021. URL https://arxiv.org/abs/2103.00564.

Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., and Li, Y. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *Trans. Recomm. Syst.*, 1(1):1–51, 2023. doi: 10.1145/3568022. URL https://doi.org/10.1145/3568022.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Ivashkin, V. and Chebotarev, P. Do logarithmic proximity measures outperform plain ones in graph clustering? In *International Conference on Network Analysis*, pp. 87–105. Springer, 2016.

Kim, H., Lee, B. S., Shin, W., and Lim, S. Graph anomaly detection with graph neural networks: Current status and challenges. *IEEE Access*, 10:111820–111829, 2022. doi: 10.1109/ACCESS.2022.3211306. URL https://doi.org/10.1109/ACCESS.2022.3211306.

Kim, H., Kim, J., Lee, B. S., and Lim, S. Label-based graph augmentation with metapath for graph anomaly detection. *Expert Syst. Appl.*, 296:129087, 2026. doi: 10.1016/J.ESWA.2025.129087. URL https://doi.org/10.1016/j.eswa.2025.129087.

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.

Kriege, N. M., Johansson, F. D., and Morris, C. A survey on graph kernels. *Applied Network Science*, 5 (1), January 2020. ISSN 2364-8228. doi: 10.1007/s41109-019-0195-3. URL http://dx.doi.org/10.1007/s41109-019-0195-3.

Li, J., Gao, Y., Lu, J., Fang, J., Wen, C., Lin, H., and Wang, X. Diffgad: A diffusion-based unsupervised graph anomaly detector. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=AhcYq4CnfF.

Liu, Y., Ding, K., Lu, Q., Li, F., Zhang, L. Y., and Pan, S. Towards self-interpretable graph-level anomaly detection. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS*

2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

Mao, Z., Li, J., Liang, C., Das, D., Sumita, M., and Tsuda, K. Molecule graph networks with many-body equivariant interactions. *CoRR*, abs/2406.13265, 2024. doi: 10.48550/ARXIV.2406.13265. URL https://doi.org/10.48550/arXiv.2406.13265.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, November 2021. ISSN 1076-9757. doi: 10.1613/jair.1.13225. URL http://dx.doi.org/10.1613/jair.1.13225.

Noble, C. C. and Cook, D. J. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 631–636, 2003.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Reid, I., Weller, A., and Choromanski, K. M. Quasi-monte carlo graph random features. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Reid, I., Berger, E., Choromanski, K. M., and Weller, A. Repelling random walks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=31IOmrnoP4.

Reid, I., Choromanski, K. M., Berger, E., and Weller, A. General graph random features. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=viftsX50Rt.

Reid, I., Dubey, K. A., Jain, D., Whitney, W. F., Ahmed, A., Ainslie, J., Bewley, A., Jacob, M. G., Mehta, A., Rendleman, D., Schenck, C., Turner, R. E., Wagner, R., Weller, A., and Choromanski, K. M. Linear transformer topological masking with graph random features. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=6MBqQLp17E.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., and Borgwardt, K. M. Efficient graphlet kernels for large graph comparison. In Dyk, D. A. V. and Welling, M. (eds.), *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pp. 488–495. JMLR.org, 2009. URL http://proceedings.mlr.press/v5/shervashidze09a.html.

Smola, A. J. and Kondor, R. Kernels and regularization on graphs. In Schölkopf, B. and Warmuth, M. K. (eds.), *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, volume 2777 of *Lecture Notes in Computer Science*, pp. 144–158. Springer, 2003. doi: 10.1007/978-3-540-45167-9_12. URL https://doi.org/10.1007/978-3-540-45167-9_12.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *The Journal of Machine Learning Research*, 11:1201–1242, 2010.

Wu, L., Yen, I. E., Zhang, Z., Xu, K., Zhao, L., Peng, X., Xia, Y., and Aggarwal, C. C. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., and Karypis, G. (eds.), *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pp. 1418–1428. ACM, 2019. doi: 10.1145/3292500.3330918. URL https://doi.org/10.1145/3292500.3330918.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yanardag, P. and Vishwanathan, S. V. N. Deep graph kernels. In Cao, L., Zhang, C., Joachims, T., Webb, G. I., Margineantu, D. D., and Williams, G. (eds.), *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pp. 1365–1374. ACM, 2015. doi: 10.1145/2783258.2783417. URL https://doi.org/10.1145/2783258.2783417.

Yang, H., Zhao, X., Li, Y., Chen, H., and Xu, G. An empirical study towards prompt-tuning for graph contrastive pre-training in recommendations. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Yang, X., Li, X., Chang, H., Yang, J., Yang, X., Tao, S., Chang, N., Shigeno, M., Wang, J., Yin, D., and Min, E. Hgformer: Hyperbolic graph transformer for recommendation. *ICML 2025*, abs/2502.15693, 2025. doi: 10.48550/ARXIV.2502.15693. URL https://doi.org/10.48550/arXiv.2502.15693.

Yannick-S. geometric_shapes: Representation of geometric shapes. https://github.com/Yannick-S/geometric_shapes, 2025. Commit snapshot as of May 28, 2026.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11782. URL https://ojs.aaai.org/index.php/AAAI/article/view/11782.

Zhang, Y., Wei, Z., Yuan, Y., Li, C., and Huang, W. Equipocket: an e(3)-equivariant geometric graph neural network for ligand binding site prediction. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=1vGN3CSxVs.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Zhou, Q. and Jacobson, A. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

A. APPENDIX

A.1. Proof of Lemma 3.1

Proof. By using similar analysis, as in the proof of Theorem 2.1 from (Reid et al., 2024b), we obtain:

$$\mathbb{E}[\mathbf{K}_1^{(i)}\mathbf{K}_2^{(i)}(a,b)] = \sum_v \sum_{p=0}^{\infty}\sum_{t=0}^{\infty} \mathbf{W}^p(a,v)\mathbf{W}^t(v,b)f(p)f(t)\mathbb{P}[X \geq p]\mathbb{P}[Y \geq t]\frac{1}{\tau(p)}\frac{1}{\tau(t)}, \tag{10}$$

where $X, Y \overset{\text{iid}}{\sim} \mathbf{P}$. Thus, form the fact that pairs $(\mathbf{K}_1^{(i)}, \mathbf{K}_2^{(i)})$ are constructed independently for different $i = 1, ..., l$, we obtain:

$$\begin{aligned}\mathbb{E}\left[\prod_{i=1}^{l}\mathbf{K}_1^{(i)}\mathbf{K}^{(i)}(a,b)\right] = \sum_{v_1,v_2,...v_{2l-1}} \sum_{p^{(1)}}^{\infty}...\sum_{p^{(l)}}^{\infty}\sum_{t^{(1)}}^{\infty}...\sum_{t^{(l)}}^{\infty} \mathbf{W}^{p^{(1)}}(a,v_1)\mathbf{W}^{t^{(1)}}(v_1,v_2)... \\ \mathbf{W}^{p^{(l)}}(v_{2l-2},v_{2l-1})\mathbf{W}^{t^{(l)}}(v_{2l-1},b)f(p^{(1)})f(t^{(1)})...f(p^{(l)})f(t^{(l)})\end{aligned} \tag{11}$$

Therefore, we obtain:

$$\mathbb{E}\left[\prod_{i=1}^{l}\mathbf{K}_1^{(i)}\mathbf{K}^{(i)}\right] = \sum_{i=0}^{\infty}\left[\sum_{p^{(1)}+t^{(1)}+...+p^{(l)}+t^{(l)}=i} f(p^{(1)})f(t^{(1)})...f(p^{(l)})f(t^{(l)})\right]\mathbf{W}^i \tag{12}$$

That completes the proof, because of Equation 4. □

A.2. Proof of Lemma 3.2

We now provide the proof of Lemma 3.2, that we re-state here for reader's convenience:

Lemma A.1 (MSE of the GRFs++ based graph estimator with GRFs++ degree $l = 2$)**.** *The* MSE *of the estimator* $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})$ *of the groundtruth graph kernel matrix* $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$*, leveraging GRFs++ with degree* $l = 2$ *satisfies (proof in the Appendix):*

$$\mathrm{MSE}(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}(\mathbf{W})) \stackrel{\text{def}}{=} \mathbb{E}[\|\mathbf{X}_1\mathbf{X}_2 - \mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2] = \|\mathbb{E}[\mathbf{X}_1^\top\mathbf{X}_1]\|_{\mathrm{F}}^2 - \|\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2 \tag{13}$$

Proof. We have the following:

$$\begin{aligned}\mathbb{E}[\|\mathbf{X}_1\mathbf{X}_2 - \mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2] = \mathbb{E}[\|\mathbf{X}_1\mathbf{X}_2 - \mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2] = \mathbb{E}[\|\mathbf{X}_1\mathbf{X}_2\|_{\mathrm{F}}^2] - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \\ \mathbb{E}[\mathrm{tr}((\mathbf{X}_1\mathbf{X}_2)(\mathbf{X}_1\mathbf{X}_2)^\top)] - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \mathbb{E}[\mathrm{tr}(\mathbf{X}_1\mathbf{X}_2\mathbf{X}_2^\top\mathbf{X}_1^\top)] - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \\ \mathbb{E}[\mathrm{tr}(\mathbf{X}_1^\top\mathbf{X}_1\mathbf{X}_2\mathbf{X}_2^\top)] - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \mathrm{tr}(\mathbb{E}[\mathbf{X}_1^\top\mathbf{X}_1\mathbf{X}_2\mathbf{X}_2^\top]) - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \\ = \mathrm{tr}(\mathbb{E}[\mathbf{X}_1^\top\mathbf{X}_1]\mathbb{E}[\mathbf{X}_2\mathbf{X}_2^\top]) - \|\mathbb{E}[\mathbf{X}_1\mathbf{X}_2]\|_{\mathrm{F}}^2 = \|\mathbb{E}[\mathbf{X}_1^\top\mathbf{X}_1]\|_{\mathrm{F}}^2 - \|\mathbb{E}[\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})\|_{\mathrm{F}}^2\end{aligned} \tag{14}$$

In the series of equalities above, we applied several facts:

1. unbiasedness of the estimator: $\mathbb{E}[\mathbf{X}_1\mathbf{X}_2] = \mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})$,
2. standard formula for the scalar variance: $\mathbb{E}[(Z - \mathbb{E}[Z])^2] = \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2$, lifted to the matrix space via Frobenius norm,
3. the following formula: $\|\mathbf{Z}\|_{\mathrm{F}}^2 = \mathrm{tr}(\mathbf{Z}\mathbf{Z}^\top)$, where tr denotes *trace* of the input matrix,
4. cyclic property of the trace: $\mathrm{tr}(\mathbf{ABCD}) = \mathrm{tr}(\mathbf{BCDA})$,
5. the symmetry of $\mathbf{X}_1^\top\mathbf{X}_1$ and $\mathbf{X}_2\mathbf{X}_2^\top$,
6. the equality: $\mathbb{E}[\mathbf{X}_1^\top\mathbf{X}_1] = \mathbb{E}[\mathbf{X}_2\mathbf{X}_2^\top]$ that comes from the definition of $\mathbf{X}_1$ and $\mathbf{X}_2$,
7. independence of $\mathbf{X}_1$ and $\mathbf{X}_2$.

□

A.3. Proof of Theorem 3.3

We will now provide a proof of Theorem 3.3.

Proof. Without loss of generality, we will assume that $m = 1$. Take two vertices: i and j of a fixed graph G. We will consider two GRFs++ estimators of the value $\mathbf{K}_{\boldsymbol{\alpha}}(\mathbf{W})[i,j]$ of the graph kernel between them: $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^t)}(\mathbf{W})[i,j]$ and $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^{t+1})}(\mathbf{W})[i,j]$, applying GRFs++ mechanism with degree 2^t and 2^{t+1} respectively (for $t \geq 0$). Note that since both estimators are unbiased, it only suffices to prove the following:

$$\mathbb{E}[(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^{t+1})}(\mathbf{W})[i,j])^2] \leq \mathbb{E}[(\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^t)}(\mathbf{W})[i,j])^2] \tag{15}$$

Note that estimator $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^t)}(\mathbf{W})[i,j]$ can be re-written as:

$$\sum_{\substack{\omega \in \Omega(i,j) \\ i=p_0,v_1,p_1,...,v_{2^t},p_{2^t}=j}} X_{f^{(2^t)}}^{(\omega)}(p_0,v_1)...X_{f^{(2^t)}}^{(\omega)}(p_{2^t-1},v_{2^l})X_{f^{(2^t)}}^{(\omega)}(p_1,v_1)...X_{f^{(2^t)}}^{(\omega)}(p_{2^t},v_{2^l}), \tag{16}$$

where:

1. $\Omega(i,j)$ is the set of all the walks between i and j
2. $p_0,...,p_{2^t}$ are some vertices (potentially with repetitions) from ω, visited in that order along the walk ω, as going from i to j
3. $X_f^{(\omega)}(a,b)$ is a random variable that is equal to

$$f(l(\omega(a,b)))W(a,b)\left(\prod_{v \in \omega(a,b)} \deg(v)\right)(1-p_{\text{halt}})^{-l(\omega(a,b))}$$

   (for $\omega(a,b)$ denoting vertices on ω from a, but not including b, $W(a,b)$ denoting the corresponding product of edge weights and $l(\omega(a,b))$ being the number of edges of the part of ω from a to b) if a random walk from a reaches b, as a prefix of ω starting from a and going to b and is zero otherwise.
4. $f^{(l)}$ is a modulation function for the GRFs++ mechanism of degree l.

Similarly, one can write $\widehat{\mathbf{K}}_{\boldsymbol{\alpha}}^{(2^{t+1})}(\mathbf{W})[i,j]$ as:

$$\begin{aligned}\sum_{\substack{\omega \in \Omega(i,j) \\ i=p_0,v_1,p_1,...,v_{2^t},p_{2^t}=j \\ u_1,u_2,...,u_{2^{t+1}-1},u_{2^{t+1}}}} & X_{f^{(2^{t+1})}}^{(\omega)}(p_0,v_1)...X_{f^{(2^{t+1})}}^{(\omega)}(p_{2^t-1},v_{2^t})X_{f^{(2^{t+1})}}^{(\omega)}(p_1,v_1)...X_{f^{(2^{t+1})}}^{(\omega)}(p_{2^t},v_{2^t}) \\ & X_{f^{(2^{t+1})}}^{(\omega)}(p_0,u_1)X_{f^{(2^{t+1})}}^{(\omega)}(p_1,u_3)...X_{f^{(2^{t+1})}}^{(\omega)}(p_{2^t-1},u_{2^{t+1}-1}) \\ & X_{f^{(2^{t+1})}}^{(\omega)}(v_1,u_1)X_{f^{(2^{t+1})}}^{(\omega)}(v_2,u_3)...X_{f^{(2^{t+1})}}^{(\omega)}(v_{2^t},u_{2^{t+1}-1}) \\ & X_{f^{(2^{t+1})}}^{(\omega)}(v_1,u_2)X_{f^{(2^{t+1})}}^{(\omega)}(v_2,u_4)...X_{f^{(2^{t+1})}}^{(\omega)}(v_{2^t},u_{2^{t+1}}) \\ & X_{f^{(2^{t+1})}}^{(\omega)}(p_1,u_2)X_{f^{(2^{t+1})}}^{(\omega)}(p_2,u_4)...X_{f^{(2^{t+1})}}^{(\omega)}(p_{2^t},u_{2^{t+1}})\end{aligned} \tag{17}$$

Denote the sub-sum of the above sum, corresponding to the particular choice of: $p_1,...,p_{2^t},v_1,...,v_{2^t}$ as: $\Psi(p_1,...,p_{2^t},v_1,...,v_{2^t})$.

It suffices to prove that for any two sequences $p_1,p_2,...,p_{2^t},v_1,...,v_{2^t}$ and $p'_1,p'_2,...,p'_{2^t},v'_1,...,v'_{2^t}$, the following holds:

$$\mathbb{E}[\Big(X^{(\omega)}_{f(2^t)}(p_0,v_1)...X^{(\omega)}_{f(2^t)}(p_{2^t-1},v_{2^l})X^{(\omega)}_{f(2^t)}(p_1,v_1)...X^{(\omega)}_{f(2^t)}(p_{2^t},v_{2^l})\Big)$$
$$\Big(X^{(\omega)}_{f(2^t)}(p'_0,v'_1)...X^{(\omega)}_{f(2^t)}(p'_{2^t-1},v'_{2^l})X^{(\omega)}_{f(2^t)}(p'_1,v'_1)...X^{(\omega)}_{f(2^t)}(p'_{2^t},v'_{2^t})\Big)] \geq \tag{18}$$
$$\mathbb{E}\left[\Psi(p_1,...,p_{2^t},v_1,...,v_{2^t})\Psi(p'_1,...,p'_{2^t},v'_1,...,v'_{2^t})\right]$$

This however follows from the convolutional properties of the modulation function f (Lemma 2.1) and the fact that the product of two X-variables corresponding to some walk starting at some fixed vertex of a graph G is not identically zero if and only if one of the walks is a prefix of another one. □

B. Additional Experimental Details

In this section, we provide additional details regarding the experimental setup and present additional results. Moreover we show a plot (Fig. 6) highlighting the speed gains by GRF++ over the baseline kernel. At moderate mesh sizes (2K–4K), GRF++ gives $\sim 1.7\times - 2.2\times$ speedup while for larger meshes (6K–10K), GRF++ consistently delivers $\sim 3\times$ faster runtime. Thus GRF++ scales significantly better as mesh complexity increases.

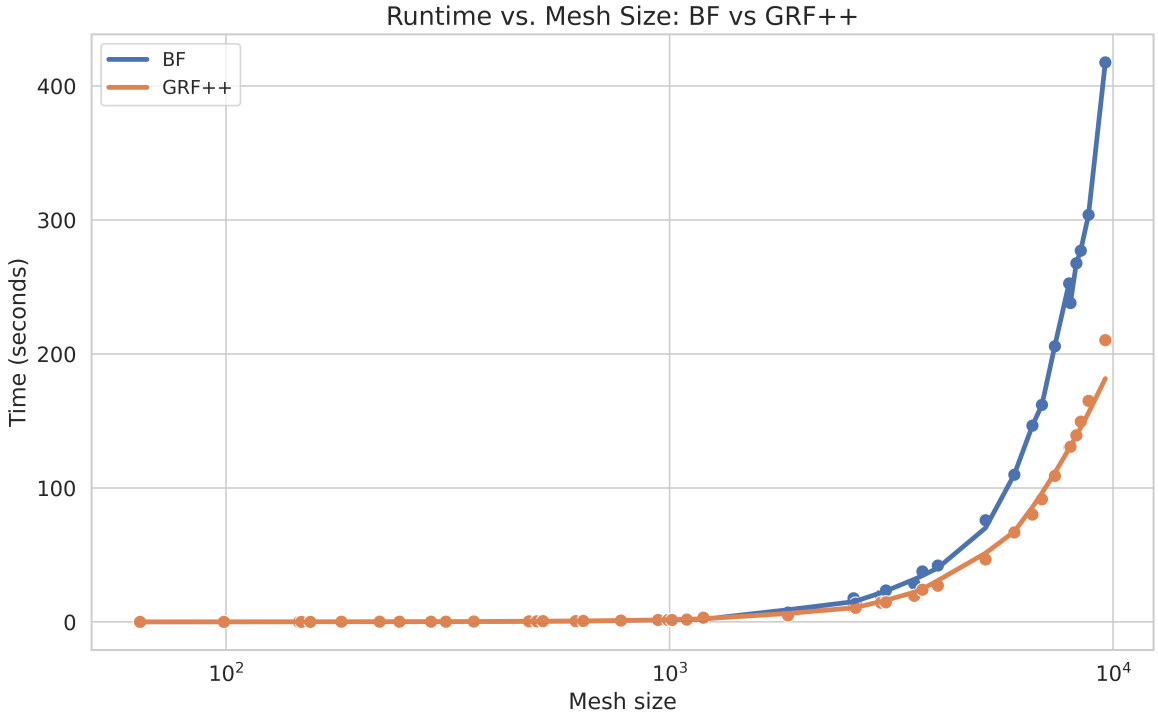

Figure 6. Time comparison between GRF++ and baseline diffusion kernel (BF) over various mesh sizes. GRF++ is significantly faster than the baseline as the graphs get larger.

B.1. Accurate estimation of Graph Kernels

We follow the exact setup as (Reid et al., 2024b). For computational comparison we used a randomly generated connected graph with 500 nodes. To have fair comparison we derived the relevant $p_{halt} = p_{base} * degree - of - kernel$. We fixed the number of random walks to 256.

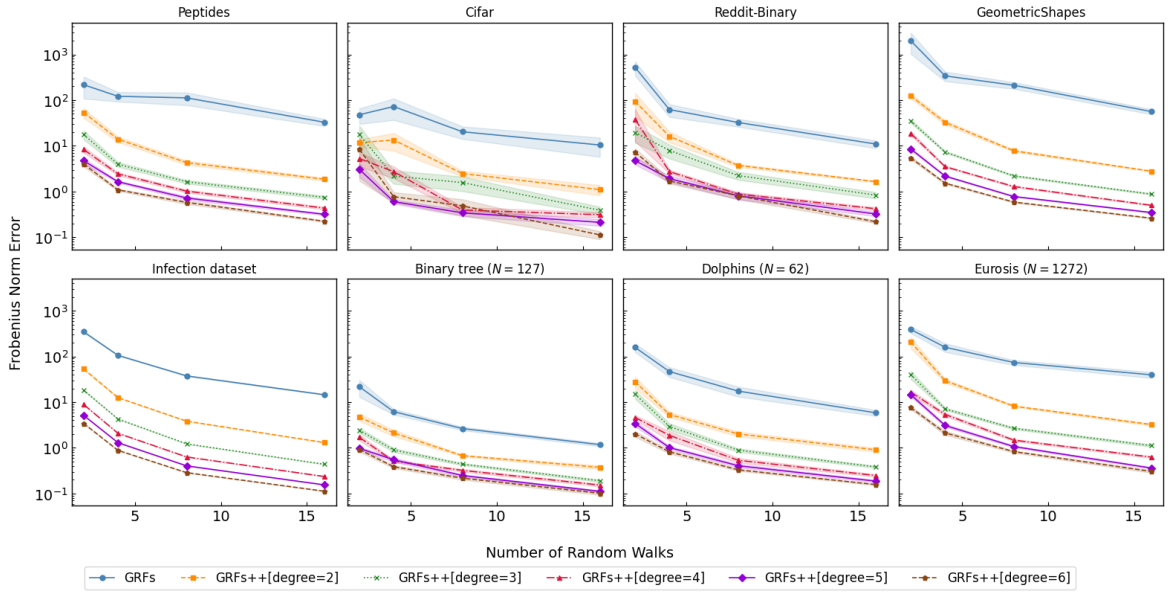

Figure 7. Estimation of the kernel values for distant nodes for the diffusion kernel. GRF++ provides a more accurate estimation in various graphs of large diameters.

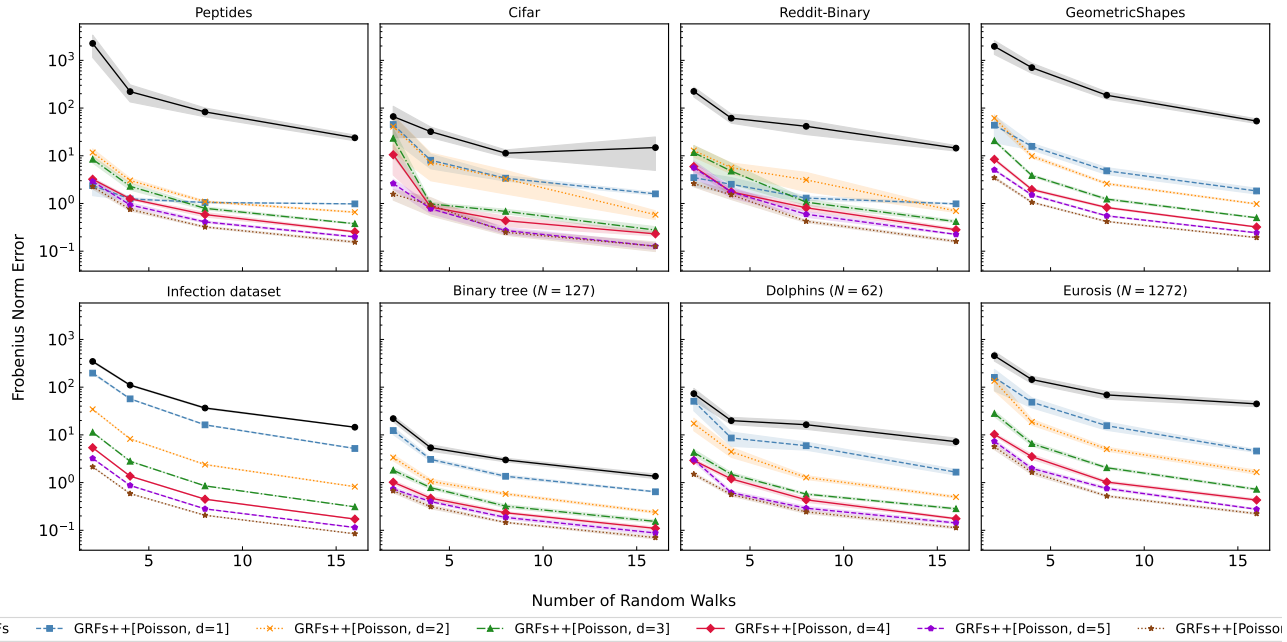

Figure 8. As in Fig. 7, but with Poisson termination strategy activated. This novel halting strategy, proposed in this paper, further improves approximation quality.

B.1.1. Experiments on Graphs with Large Diameters

In this subsection, we provide details on the graphs used for estimating longer walks. For this task, we pick graphs from Peptides (Dwivedi et al., 2022), CIFAR-10 (Dwivedi et al., 2023), Reddit-Binary (Morris et al., 2020), Geometric Shapes (Yannick-S, 2025) as well from the dataset considered by Reid et al. (2024b).

For each of these datasets, we remove isolated nodes and select the graphs with the largest diameters from the subset of the connected graphs. Finally to threshold the graphs to select the longer walks, we compute the shortest path distance via the Floyd-Warshall algorithm. We then select the pair of nodes where the walks are longer than the specified distance away. We

then use this information to mask (i.e. zero out) all entries in the diffusion kernel.

We provide the walk threshold for these graphs in Table 6.

Table 6. Statistics of datasets used in experiments for estimate the accuracy to capture long walks. The walk length column refers to the fact that we are estimating all walks $\geq k$.

Dataset	Diameter	# Nodes	Walk Length (k)
Peptides	159	434	3
CIFAR	11	128	4
Reddit-Binary	19	436	5
GeometricShapes	28	864	5
Infection	4	500	4
Binary Tree	12	127	3
Dolphins	8	62	4
Eurosis	10	1272	4

B.2. Graph Classification Experiments

In this subsection we provide additional details about our graph classification experiments. The statistics of our datasets is provided in Table 7 with additional details provided in (Morris et al., 2020). We follow the framework proposed by (Errica et al., 2020) to evaluate the performance of the diffusion kernel as well as the approximate kernels obtained by GRF and GRF++. In particular, we use 10-fold cross-validation to obtain an estimate of the generalization performance of the methods.

Finally, we follow the approach by (de Lara & Pineau, 2018) to create graph features by using the smallest k-eigenvalues of the corresponding kernels. These features are then passed to a random forest classifier for classification. k is independently for the baseline as well as for GRF and GRF++.

We did a small hyperparameter sweep over $\{.5, .6, .8, .9\}$ to find the width of the diffusion kernel. For GRF and GRF++, we fix the halting probability to be $.1$ and do a hyperparameter sweep over the number of walks. The degree of GRF++ is chosen to be 2.

Finally we posit our results in the context of various kernel methods as well as GNNs. In particular, we compare our methods against 20 popular kernels like (1) Vertex Histogram kernel (VH), (2) Random Walk kernel (RW), (3) Shortest Path kernel (SP), (4) Graphlet kernel (GR), (5) Weisfeiler-Lehman sub-tree kernel (WL-VH), (6) Weisfeiler-Lehman shortest path kernel (WL-SP), (7) Weisfeiler-Lehman pyramid match kernel (WL-PM), (8) Weisfeiler-Lehman optimal assignment kernel (WLOA), (9) Neighborhood Hash kernel (NH), (10) Neighborhood subgraph pairwise distance kernel (NSPDK), (11) Lovász ϑ kernel (Lo-ϑ), (12) SVM-ϑ kernel (SVM-ϑ), (13) Ordered Decompositional DAGs with subtree kernel (ODD-STh), (14) Pyramid Match kernel (PM), (15) GraphHopper kernel (GH), (16) Subgraph Matching kernel (SM), (17) Propagation kernel (PK), (18) Multiscale Laplacian kernel (ML), (19) Core Weisfeiler-Lehman subtree kernel (CORE-WL-VH), and (20) Core Shortest Path kernel (CORE-SP). Note that this kernel computes relationship between graphs whereas our GRF++ computes relationship between nodes in a given graph. Finally we also compare our methods to popular GNNs like (1) DGCNN (Zhang et al., 2018), (2) GraphSAGE (Hamilton et al., 2017), (3) DiffPool (Ying et al., 2018), and (4) GIN (Xu et al., 2019). All these baseline numbers are taken from (Nikolentzos et al., 2021). Our results show that diffusion kernel is competitive across all the datasets. Our GRF++ mechanism maintains similar performance to that of the baseline diffusion kernel while being more computationally efficient. For unlabeled graphs, the diffusion kernel as well as GRF++ offers competitive performance across 20+ methods. The performance of the diffusion kernel as well as GRF++ dips for node labeled graphs as they do not take into account the node labels. However our algorithm can be modified to construct walks among nodes that share the same label using the ideas from (Choromanski et al., 2025). We leave this extension for future work.

B.3. Node Clustering

We follow the same setup as (Reid et al., 2024b) except that the number of clusters is based upon the actual number of different classes. Thus we use $p_{halt} = 0.1$, $m = 16$. To get details of the dataset, please see (Ivashkin & Chebotarev, 2016).

Table 7. Statistics of the graph classification datasets used in this paper.

DATASETS	# Graphs	# Labels	Avg. # Nodes	Avg. # Edges	# Node Labels	# Node Attributes
MUTAG	188	2	17.93	19.79	7	-
PTC-MR	344	2	14.29	14.69	19	-
ENZYMES	600	6	32.63	62.14	3	18
PROTEINS	1113	2	39.06	72.82	3	1
D&D	1178	2	284.32	715.66	82	-
IMDB BINARY	1000	2	19.77	96.53	-	-
IMDB MULTI	1500	3	13.0	65.94	-	-
NCI1	4110	2	29.87	32.30	37	-
COLLAB	5000	3	74.49	2457.78	-	-
REDDIT BINARY	2000	2	429.63	497.75	-	-
REDDIT MULTI-5K	4999	5	508.52	594.87	-	-
REDDIT MULTI-12K	11929	11	391.41	456.89	-	-

	Methods	DATASETS						Rank
		MUTAG	ENZYMES	NCI1	PTC-MR	D&D	PROTEINS	
Kernels	VH	69.1 (± 4.1)	20.0 (± 4.8)	55.7 (± 2.0)	57.1 (± 9.6)	74.8 (± 3.7)	71.1 (± 4.4)	21.0
	RW	81.4 (± 8.9)	16.7 (± 1.8)	TIMEOUT	54.4 (± 9.8)	OUT-OF-MEM	69.5 (± 5.1)	22.0
	SP	82.4 (± 5.5)	37.3 (± 8.7)	72.5 (± 2.0)	60.2 (± 9.4)	77.9 (± 4.5)	74.9 (± 3.6)	13.8
	WL-VH	86.7 (± 7.3)	50.7 (± 7.3)	85.2 (± 2.2)	64.9 (± 6.4)	78.7 (± 2.3)	76.2 (± 3.5)	4.6
	WL-SP	81.4 (± 8.7)	27.3 (± 7.4)	60.8 (± 2.4)	54.5 (± 9.8)	76.0 (± 3.5)	72.1 (± 3.1)	19.2
	WL-PM	88.3 (± 7.1)	57.5 (± 6.8)	85.6 (± 1.7)	65.1 (± 7.5)	OUT-OF-MEM	75.9 (± 3.8)	5.8
	WL-OA	87.2 (± 5.4)	58.0 (± 5.0)	86.3 (± 1.6)	65.7 (± 9.6)	77.6 (± 3.0)	76.2 (± 3.9)	3.2
	NH	88.3 (± 6.3)	54.5 (± 3.6)	84.7 (± 1.9)	63.4 (± 9.2)	74.6 (± 3.5)	75.0 (± 4.2)	6.6
	NSPDK	85.6 (± 8.9)	42.2 (± 8.0)	74.3 (± 2.1)	59.1 (± 7.3)	78.9 (± 4.7)	72.5 (± 2.9)	11.4
	ODD-STh	80.4 (± 8.8)	32.3 (± 4.8)	75.2 (± 2.0)	59.4 (± 9.8)	76.4 (± 4.5)	70.9 (± 4.1)	16.4
	PM	85.1 (± 5.8)	43.2 (± 5.3)	73.5 (± 1.9)	60.2 (± 8.2)	77.9 (± 3.7)	70.9 (± 4.4)	11.2
	GH	82.5 (± 5.8)	37.2 (± 6.6)	71.0 (± 2.3)	60.2 (± 9.4)	TIMEOUT	74.8 (± 2.4)	17.4
	SM	85.7 (± 5.8)	35.7 (± 5.5)	TIMEOUT	60.2 (± 6.8)	OUT-OF-MEM	OUT-OF-MEM	16.8
	PK	76.6 (± 5.2)	44.0 (± 6.3)	82.1 (± 2.1)	65.1 (± 5.6)	77.7 (± 4.2)	73.1 (± 4.7)	10.2
	ML	87.2 (± 7.5)	48.5 (± 7.8)	79.7 (± 1.8)	64.5 (± 5.8)	78.6 (± 4.0)	74.2 (± 4.4)	6.0
	CORE-WL-VH	85.6 (± 6.5)	51.7 (± 7.0)	85.2 (± 2.2)	65.5 (± 5.6)	79.5 (± 3.2)	76.5 (± 4.4)	4.0
	CORE-SP	85.1 (± 6.8)	39.5 (± 9.3)	73.8 (± 1.4)	57.3 (± 9.7)	79.3 (± 3.8)	76.5 (± 3.9)	13.0
	Diffusion	86.2 (± 4.2)	41.7 (± 3.6)	71.5 (± 1.3)	62.2 (± 5.4)	74.8 (± 2.2)	74.5 (± 2.4)	15.8
	GRF	86.7 (± 4.4)	39.8 (± 3.2)	70.0 (± 1.6)	59.7 (± 5.8)	73.4 (± 2.4)	73.7 (± 2.3)	15.2
	GRF++ (ours)	87.8 (± 5.2)	40.9 (± 3.9)	72.4 (± 1.2)	61.3 (± 5.7)	74.5 (± 2.6)	74.3 (± 2.8)	12.4
GNNs	DGCNN	84.0 (± 7.1)	46.3 (± 6.3)	76.4 (± 1.7)	59.5 (± 6.9)	76.6 (± 4.3)	73.2 (± 3.2)	11.8
	GraphSAGE	83.6 (± 9.6)	46.1 (± 5.4)	76.0 (± 1.8)	61.7 (± 4.9)	72.9 (± 2.0)	74.3 (± 3.8)	12.8
	DiffPool	79.8 (± 6.7)	50.7 (± 8.7)	76.9 (± 1.9)	61.1 (± 5.6)	75.0 (± 3.5)	72.5 (± 3.5)	12.0
	GIN	84.7 (± 6.7)	44.5 (± 4.1)	80.0 (± 1.4)	59.1 (± 7.0)	75.3 (± 2.9)	72.8 (± 3.6)	12.4

Table 8. Average classification accuracy (± standard deviation) on the 6 classification datasets containing node-labeled graphs.

In table 11, we compare GRF++ with 5 other methods : Louvain, Spectral, GRF, KCenters and Propagation. Our method is competitive across all datasets and outperforms all methods on 3 out of 5 datasets.

B.4. Vertex Normal Prediction Experiments

In this sub-section, we present implementation details for vertex normal prediction experiments. All the experiments are run on free Google Colab with 12Gb of RAM.

For this task, we choose 40 meshes for 3D-printed objects of varying sizes from the Thingi10K dataset. Following (Choromanski et al., 2024), we choose the following meshes corresponding to the ids given by :

```
[60246, 85580, 40179, 964933, 1624039, 91657, 79183, 82407, 40172, 65414, 90431,
74449, 73464, 230349, 40171, 61193, 77938, 375276, 39463, 110793, 368622, 37326,
42435, 1514901, 65282, 116878, 550964, 409624, 101902, 73410, 87602, 255172,
```

	Methods	DATASETS						Rank
		IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K	COLLAB	
Kernels	VH	50.0 (± 0.0)	33.3 (± 0.0)	50.0 (± 0.0)	20.0 (± 0.0)	21.7 (± 1.5)	52.0 (± 0.1)	21.50
	RW	64.1 (± 4.5)	44.6 (± 4.1)	TIMEOUT	TIMEOUT	TIMEOUT	68.0 (± 1.7)	22.17
	SP	58.2 (± 4.7)	39.2 (± 2.3)	81.7 (± 2.5)	47.9 (± 1.9)	TIMEOUT	58.8 (± 1.2)	19.33
	GR	66.1 (± 2.7)	39.5 (± 2.7)	76.1 (± 2.6)	34.7 (± 2.0)	23.0 (± 1.4)	73.0 (± 2.0)	15.83
	WL-VH	70.7 (± 6.8)	51.3 (± 4.4)	67.8 (± 3.5)	50.5 (± 1.6)	38.7 (± 1.7)	78.3 (± 2.1)	7.33
	WL-SP	58.2 (± 4.7)	39.2 (± 2.3)	TIMEOUT	TIMEOUT	TIMEOUT	58.8 (± 1.2)	24.17
	WL-PM	73.6 (± 3.4)	49.1 (± 5.5)	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	10.33
	WL-OA	72.6 (± 5.5)	51.1 (± 4.3)	89.0 (± 1.3)	54.0 (± 1.2)	TIMEOUT	80.5 (± 2.0)	7.17
	NH	71.6 (± 4.5)	50.5 (± 5.0)	81.2 (± 2.0)	49.9 (± 2.4)	39.6 (± 1.4)	81.1 (± 2.4)	6.67
	NSPDK	67.4 (± 3.3)	44.6 (± 3.8)	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	23.50
	Lo-ϑ	51.0 (± 4.2)	39.8 (± 2.6)	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	25.33
	SVM-ϑ	52.3 (± 4.0)	39.5 (± 2.7)	74.8 (± 2.6)	31.4 (± 1.1)	22.9 (± 0.9)	52.0 (± 0.1)	19.17
	ODD-STh	65.0 (± 4.0)	46.7 (± 3.4)	52.1 (± 3.2)	43.1 (± 1.8)	30.0 (± 1.6)	52.0 (± 0.1)	16.33
	PM	66.3 (± 4.2)	46.1 (± 3.8)	86.5 (± 2.1)	48.3 (± 2.5)	41.1 (± 0.6)	74.0 (± 2.4)	10.33
	GH	59.4 (± 3.4)	39.5 (± 2.6)	TIMEOUT	TIMEOUT	TIMEOUT	60.0 (± 1.4)	23.17
	SM	TIMEOUT	TIMEOUT	OUT-OF-MEM	OUT-OF-MEM	OUT-OF-MEM	TIMEOUT	27.00
	PK	51.7 (± 3.7)	34.5 (± 3.0)	63.9 (± 3.0)	34.9 (± 1.7)	23.9 (± 1.2)	57.0 (± 1.2)	19.17
	ML	69.9 (± 4.8)	47.7 (± 3.2)	89.4 (± 2.1)	35.4 (± 2.0)	OUT-OF-MEM	75.6 (± 1.6)	11.83
	CORE-WL-VH	73.5 (± 6.1)	51.7 (± 4.1)	73.0 (± 4.5)	51.1 (± 1.6)	40.2 (± 1.8)	84.5 (± 2.0)	5.00
	CORE-SP	68.5 (± 3.9)	51.0 (± 3.5)	91.0 (± 1.8)	TIMEOUT	OUT-OF-MEM	TIMEOUT	16.50
	Diffusion	72.3 (± 1.8)	50.8 (± 2.0)	85.6 (± 1.2)	49.5 (± 0.4)	35.9 (± 0.7)	75.1 (± 0.2)	7.00
	GRF	68.9 (± 1.9)	48.7 (± 2.4)	84.0 (± 1.1)	48.6 (± 0.7)	35.1 (± 0.9)	70.8 (± 0.6)	11.00
	GRF++ (ours)	71.2 (± 2.1)	49.1 (± 2.9)	84.4 (± 1.5)	49.2 (± 0.9)	36.8 (± 0.9)	73.6 (± 0.9)	8.83
GNNs	DGCNN	69.2 (± 3.0)	45.6 (± 3.4)	87.8 (± 2.5)	49.2 (± 1.2)	43.9 (± 1.0)	71.2 (± 1.9)	9.50
	GraphSAGE	68.8 (± 4.5)	47.6 (± 3.5)	84.3 (± 1.9)	50.0 (± 1.3)	43.5 (± 1.0)	73.9 (± 1.7)	8.83
	DiffPool	68.4 (± 3.3)	45.6 (± 3.4)	89.1 (± 1.6)	53.8 (± 1.4)	44.4 (± 1.4)	68.9 (± 2.0)	8.83
	GIN	71.2 (± 3.9)	48.5 (± 3.3)	89.9 (± 1.9)	56.1 (± 1.7)	48.3 (± 1.6)	75.6 (± 2.3)	4.50

Table 9. Average classification accuracy (± standard deviation) on the 6 classification datasets containing unlabeled graphs.

98480, 57140, 285606, 96123, 203289, 87601, 409629, 37384, 57084]

We do a small search for the width of the kernel $\sigma \in \{.5, .6, .8\}$ for the baseline runs. For both GRF and GRF++, the number of walks are chosen from the subset $\{4, 8, 16\}$ and the halting probability of the walk is $.1$. The degree of GRF++ is chosen to be 2. We also compare against Al Mohy as it is a widely popular algorithm to compute the action of matrix exponential on a vector. We use the SciPy implementation of Al Mohy.

B.5. Vision Transformer Attention Masking

In this experiment, we investigate the utility of GRFs++ for introducing inductive biases into the self-attention mechanism of Vision Transformers (ViTs). We treat the input image patches as nodes in a regular 2D grid graph (lattice), where edges connect spatially adjacent patches. We employ GRFs++ (with degree $l = 2$) to efficiently approximate the diffusion kernel matrix K_{diff} on this grid graph. This approximate kernel is subsequently normalized and utilized as a soft mask M, which is added to the standard attention logic:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \lambda \mathbf{M}\right)\mathbf{V} \tag{19}$$

We choose $\lambda = 0.5$ in our setup. We test our model two large scale datasets : ImageNet (Deng et al., 2009) and Places365 (Zhou et al., 2017).

B.6. GRF++ beyond Diffusion Kernels

In this section, we showcase the ability of GRFs++ to efficiently approximate graph node kernels beyond the diffusion kernel. We consider the following kernels: (1) $\mathbf{K}(\mathbf{W}) = (\mathbf{I} - \mathbf{W})^{-4}$, (2) $\mathbf{K}(\mathbf{W}) = (\mathbf{I} - \mathbf{W}^2)^{-4}$, (3) $\mathbf{K}(\mathbf{W}) = \exp(\mathbf{W}^2)$,

Table 10. Cosine Similarity for Meshes. GRF++ matches the performance of the baseline kernel (BF). We also compare against Al Mohy's algorithm to compute the action of the matrix exponential on a vector. GRF++r reuses the same walk and still outperform GRF.

MESH SIZE	64	99	146	148	155	182	222	246	290	313
BF	0.4255	0.7675	0.9424	0.4325	0.7095	0.9654	0.8715	0.7464	0.8895	0.5514
Al Mohy	0.4248	0.7672	0.9420	0.4320	0.7091	0.9573	0.8704	0.7363	0.8789	0.5511
GRF	0.3083	0.6786	0.9348	0.3813	0.6831	0.9466	0.8569	0.6722	0.8651	0.5309
GRF++	0.3889	0.7163	0.9367	0.4434	0.6892	0.9611	0.8684	0.7377	0.8679	0.5432
GRF++r	0.3905	0.7163	0.9327	0.4435	0.6867	0.9564	0.8510	0.6878	0.8464	0.5178
MESH SIZE	362	482	502	518	614	639	777	942	992	1012
BF	0.5884	0.9830	0.8881	0.4956	0.9172	0.8958	0.8022	0.8559	0.7206	0.9366
Al Mohy	0.5866	0.9829	0.8872	0.4912	0.9146	0.8871	0.8009	0.8472	0.7184	0.9183
GRF	0.5751	0.9737	0.8673	0.4486	0.8866	0.8739	0.7812	0.8369	0.6967	0.9136
GRF++	0.5821	0.9807	0.8843	0.4830	0.9084	0.8914	0.8039	0.8496	0.7144	0.9234
GRF++r	0.5688	0.9775	0.8772	0.4734	0.8982	0.8866	0.8019	0.8406	0.7085	0.9200
MESH SIZE	1094	1192	1849	2599	2626	2996	3072	3559	3715	4025
BF	0.9236	0.8297	0.9265	0.4987	0.8927	0.9326	0.4796	0.9356	0.9619	0.9669
Al Mohy	0.9207	0.8283	0.9181	0.4870	0.8916	0.9236	0.4762	0.9312	0.9582	0.9599
GRF	0.9001	0.7987	0.9101	0.4065	0.8778	0.9171	0.4637	0.9208	0.9508	0.9588
GRF++	0.9170	0.8167	0.9202	0.4615	0.8820	0.9277	0.4731	0.9293	0.9546	0.9646
GRF++r	0.9098	0.8134	0.9146	0.4209	0.8730	0.9210	0.4606	0.9281	0.9561	0.9620
MESH SIZE	5155	5985	6577	6911	7386	7953	8011	8261	8449	8800
BF	0.9011	0.9194	0.9622	0.9769	0.9437	0.9460	0.9382	0.9196	0.9276	0.9836
Al Mohy	0.8907	0.9162	0.9512	0.9684	0.9420	0.9360	0.9226	0.9052	0.9163	0.9798
GRF	0.8833	0.9091	0.9525	0.9682	0.9308	0.9383	0.9233	0.9050	0.9139	0.9778
GRF++	0.8931	0.9154	0.9599	0.9751	0.9374	0.9429	0.9321	0.9145	0.9205	0.9820
GRF++r	0.8896	0.9129	0.9561	0.9701	0.9348	0.9410	0.9269	0.9095	0.9160	0.9805

Table 11. Node Clustering: Comparing GRFs++ against various clustering algorithms. Our method outperforms all other baselines on 3 out of 5 datasets.

Name	# Nodes	# clusters	Propagation	KCenters	Spectral	Louvain	GRF	GRF++[d=2]
Karate	34	2	0.4011	0.2585	0.2585	0.4367	0.2995	**0.2585**
Dolphins	62	2	0.0936	0.0323	0.0408	0.3432	0.0635	**0.0323**
Polbooks	105	3	**0.0621**	0.0621	0.0692	0.1154	0.1060	0.1033
Football	115	12	0.1986	0.0477	0.0551	**0.0174**	0.0731	0.0362
Databases	1006	6	0.4762	0.4918	0.3214	0.5298	0.3528	**0.3001**
Eurosis	1272	13	0.7418	0.1891	0.1295	0.1599	0.2248	**0.1304**

with the corresponding modulation functions being given by Taylor-series expansions of the following matrix functions: (1) $\mathbf{W} \rightarrow (\mathbf{I} - \mathbf{W})^{-1}$, (2) $\mathbf{W} \rightarrow (\mathbf{I} - \mathbf{W}^2)^{-1}$, $\mathbf{W} \rightarrow \exp(\frac{\mathbf{W}^2}{4})$.The results are presented in Fig. 9,10,11. GRFs++ outperform regular GRFs for all three kernels.

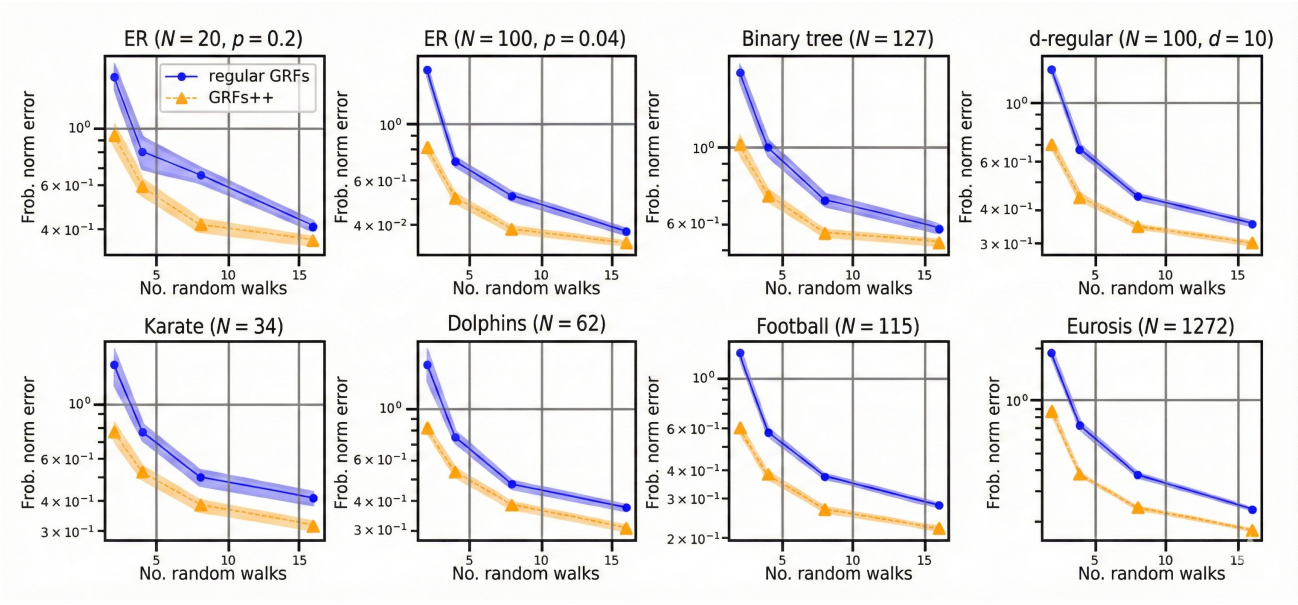

Figure 9. Experiment analogous to this from Fig. 2, but with a fixed degree $l = 2$ used in GRfs++ and graph kernel of the form $\mathbf{K}(\mathbf{W}) = (\mathbf{I} - \mathbf{W})^{-4}$.

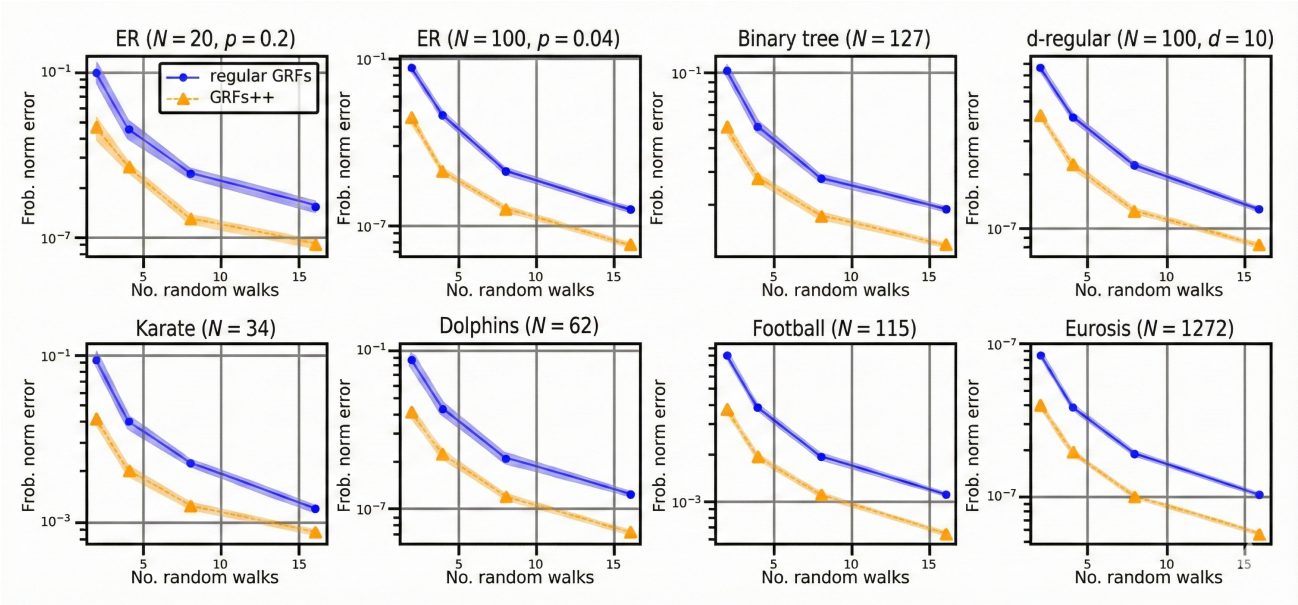

Figure 10. Experiment analogous to this from Fig. 2, but with a fixed degree $l = 2$ used in GRfs++ and graph kernel of the form $\mathbf{K}(\mathbf{W}) = (\mathbf{I} - \mathbf{W}^2)^{-4}$.

Moreover we use $K(W) = exp(W^2)$ for graph classification to show that utility of GRF++ extends beyond diffusion kernels. For all the 8 datasets, GRF++ outperforms GRF and matches the baseline.

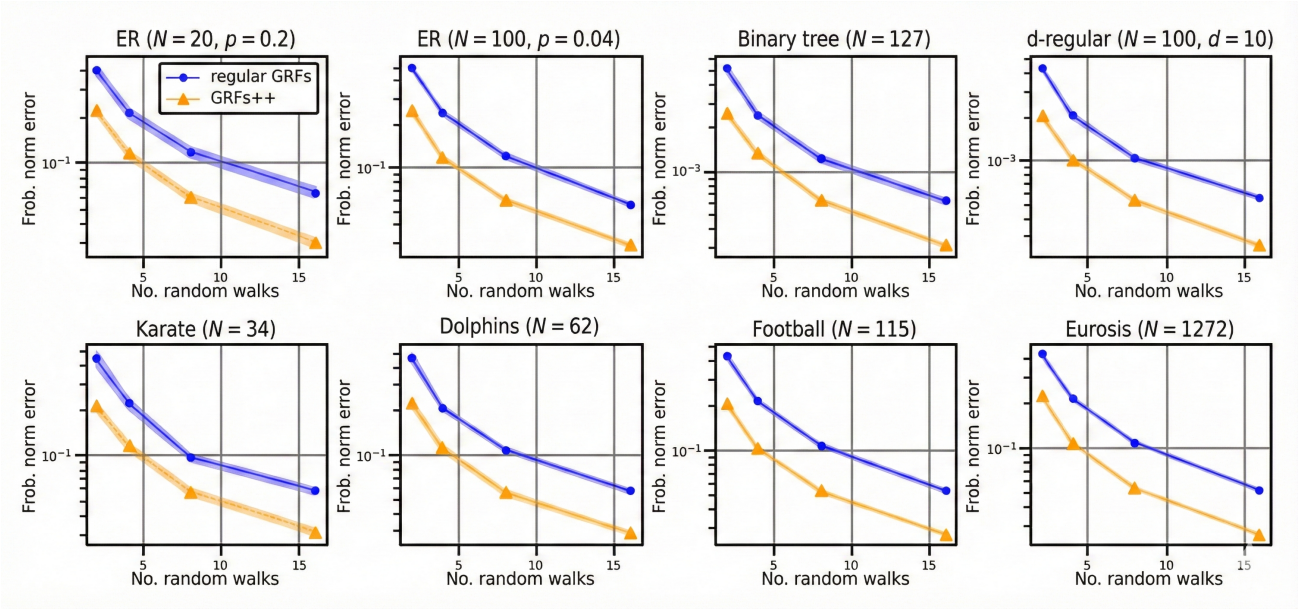

Figure 11. Experiment analogous to this from Fig. 2, but with a fixed degree $l = 2$ used in GRfs++ and graph kernel of the form $\mathbf{K}(\mathbf{W}) = \exp(\mathbf{W}^2)$.

Table 12. **Graph classification accuracy (%) across benchmark datasets.** We compare the baseline kernel with the proposed GRF and GRF++ kernels. Best results for each dataset are shown in bold.

Dataset	Baseline	GRF++	GRF
MUTAG	**86.20**	85.12	83.55
PROTEINS	74.48	**74.87**	73.16
ENZYMES	**41.67**	40.67	38.33
IMDB-BINARY	**72.30**	70.68	69.12
IMDB-MULTI	**50.82**	49.80	48.33
NCI-1	71.45	**75.47**	71.64
PTC-MR	**62.21**	61.87	59.37
DD	**74.79**	73.67	72.71

C. GRF++ in Graph GPS

The GraphGPS architecture combines local message passing with global transformer attention in order to capture both short-range and long-range dependencies in graphs. Since transformer attention itself is permutation invariant, GraphGPS augments node representations with positional and structural encodings that provide information about the location of nodes within the graph topology. Two commonly used encodings are LapPE and RWSE. LapPE uses eigenvectors of the graph Laplacian to encode global structural information, providing a spectral notion of node position, while RWSE encodes structural context through random walk return probabilities across multiple diffusion steps. These encodings are injected into the node features before attention and play a critical role in enabling the transformer layers to distinguish structurally different nodes.

In our setting, the features generated by GRF++ can naturally serve the same purpose. Rather than relying on spectral decompositions or predefined random walk statistics, GRF++ produces randomized graph features that capture multi-scale structural relationships in an efficient and flexible manner.

For the rest of the section let $d = 2$, this will give rise to 2 sets of sparse feature vectors. We use a randomized projection to create a 8-dim vector from each of these representation. To preserve invariance, we add these representations to create a 8-dimensional representation for each graph. The rest of the training details follow (Rampášek et al., 2022).

D. Reproducibility statement

The paper provides a clear description of the GRFs++ algorithm. In Sec. 2.1, we present detailed description of the regular GRFs algorithm, namely Algorithm 1 box, that GRFs++ build on. This algorithm is also implemented in the github repository mentioned on the first page of (Reid et al., 2024b). In Lemma 2.1, we explain how Algorithm 1 is used in GRFs++ for the general walk-stitching mechanism. Then in Sec. 2.2.2, we provide detailed explanation of the modification of Algorithm 1 that needs to be conducted in order to support arbitrary termination strategies (points 1-3). For all the experiments, we provided the names of all datasets and graphs used (or exact procedures to construct those graphs, e.g. random Erdős-Rényi graph models with explicitly given probabilities p of edge sampling). All the theoretical statements have all the assumptions clearly stated and the corresponding proofs given (see: Section 2.2.1, Section 3 and Appendix: Section A.1, Section A.2 and Section A.3). Finally, we open source our implementation at `https://github.com/arijitthegame/super_graph_random_features`