Graph Positional Encoding via Random Feature Propagation

Moshe Eliasof [1] **Fabrizio Frasca** [2] **Beatrice Bevilacqua** [3] **Eran Treister** [1] **Gal Chechik** [4 5] **Haggai Maron** [5]

Abstract

Two main families of node feature augmentation schemes have been explored for enhancing GNNs: random features and spectral positional encoding. Surprisingly, however, there is still no clear understanding of the relation between these two augmentation schemes. Here we propose a novel family of positional encoding schemes which draws a link between the above two approaches and improves over both. The new approach, named Random Feature Propagation (RFP), is inspired by the power iteration method and its generalizations. It concatenates several intermediate steps of an iterative algorithm for computing the dominant eigenvectors of a propagation matrix, starting from random node features. Notably, these propagation steps are based on graph-dependent propagation operators that can be either predefined or learned. We explore the theoretical and empirical benefits of RFP. First, we provide theoretical justifications for using random features, for incorporating early propagation steps, and for using multiple random initializations. Then, we empirically demonstrate that RFP significantly outperforms both spectral PE and random features in multiple node classification and graph classification benchmarks.

1. Introduction

GNN architectures such as Message-Passing Neural Networks (MPNNs) became very popular for learning with graph-structured data, but they suffer from various limitations. Primarily, they were shown to have limited expressiveness, which hurts their performance in practice (Morris et al., 2019; Xu et al., 2019; Morris et al., 2021; Zhang et al., 2021). Many approaches have been proposed to alleviate these limitations, including two prominent directions that are based on augmenting the input node features of a given graph, namely, random and spectral features. The first approach builds on random node features (RNF) (Abboud et al., 2020; Sato et al., 2021), and was shown to improve the expressiveness of MPNNs by increasing their capability to disambiguate nodes. Unfortunately, even though RNF theoretically add expressiveness, they do not consistently improve performance on real-world datasets (Bevilacqua et al., 2022). A second approach to alleviate the expressivity limitation suggests enhancing input node features with spectral PE often derived from a partial eigendecomposition of the graph-Laplacian matrix (Dwivedi & Bresson, 2021; Kreuzer et al., 2021; Wang et al., 2022; Lim et al., 2023; Rampasek et al., 2022). These methods have demonstrated an improvement on graph classification benchmarks compared to using only the raw input node features.

Our approach. This work is based on the observation that iterative matrix eigenproblem solvers provide a natural link between random node features and existing spectral PE methods. Prominent examples of these iterative algorithms include the well-known power method and its generalizations, such as subspace iteration methods (Saad, 2011). Specifically, these eigensolvers take random vectors and transform them into eigenvectors by performing a series of propagation and normalization steps. Importantly, when the input matrix represents either the graph adjacency or the Laplacian matrix, the output of these algorithms is the spectral PE mentioned above.

In this paper, we propose a family of PE schemes for graph learning, which we call *Random Feature Propagation* (RFP), designed to improve over RNF and eigendecomposition-based PE schemes while retaining the advantages of both approaches. Given a graph with n nodes, an $n \times n$ propagation matrix (e.g., the adjacency matrix), and a normalization function (e.g., ℓ_2 normalization or orthonormalization), the RFP PE is defined as a concatenation of multiple successive propagation-and-normalization steps, starting from random node features. We refer to the output of this process as the *RFP trajectory*. As we show in Section 3.1, this approach can be interpreted as an interpolation between the RNF and the spectral PE strategies outlined above. Therefore, RFP provides GNNs with features ranging from pure RNF to

[1]Department of Computer Science, Ben-Gurion University of the Negev, Israel. [2]Department of Computing, Imperial College London, UK. [3]Department of Computer Science, Purdue University, USA. [4]Department of Computer Science, Bar-Ilan University, Israel. [5]NVIDIA Research. Correspondence to: Moshe Eliasof <eliasof@post.bgu.ac.il>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

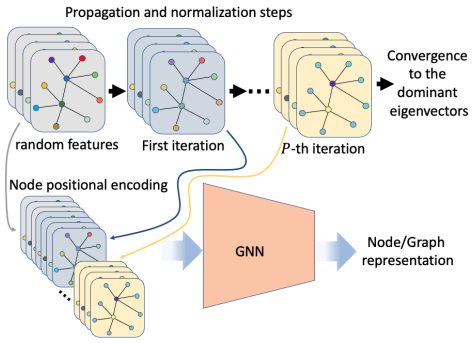

Figure 1. An overview of our Random Feature Propagation (RFP). We start from k-dimensional random node features and perform a series of propagation and normalization steps according to a predefined or learned propagation operator. We then use the whole trajectory as node PE. As a result of this process, the network has access to features ranging from pure random features to features that approximate the dominant k eigenvectors of the propagation operator. We also suggest methods for learning the propagation operator and for processing multiple trajectories.

approximations of the dominant eigenvectors of the propagation operator. Figure 1 illustrate this concept. We show empirically and theoretically that RFP combines the benefits of both approaches. In particular, we demonstrate that feeding the GNN with the RFP trajectory is more effective than feeding it with RNF and/or the eigenvectors only.

A key component of our framework is the propagation operator. We propose two types of propagation operators: the first kind is a pre-defined operator, such as the adjacency or Laplacian matrices. The second kind of operators are *learnable*, data-dependent, propagation operators that are learned jointly with the GNN in an end-to-end manner according to the downstream task. Learning the operator is performed by incorporating an attention mechanism that outputs scores for pairs of nodes. In contrast to standard propagation operators, the learned one can model additional node interactions, including long-range pairwise dependencies.

We demonstrate that RFP can be used successfully with standard GNNs such as GIN (Xu et al., 2019) and GCN (Kipf & Welling, 2017). Furthermore, we propose using *multiple* RFP trajectories as PE and processing them with a permutation equivariant architecture called DSS-GNN, which was recently suggested (Bevilacqua et al., 2022) for other purposes. The combination of multiple RFP trajectories with DSS-GNN further improves the performance.

We present several theoretical results to support the main choices made in our framework. As a first result, we show that early stages in the RFP trajectory are useful: we prove that MPNNs can use these early-stage features in order to extract structural information, such as the number of triangles in the graph. Our second result provides theoretical support for starting the propagation steps from random features rather than the original node features, as suggested in recent works (Frasca et al., 2020; Huang et al., 2022). In particular, we show that unlike in these previous works, our trajectories converge almost surely to any chosen number of dominant eigenvectors of the propagation operator. As a result, we are able to apply our method to graphs whose input feature matrices have low rank or dimensionality, such as featureless graphs. Third, our framework trivially inherits the strong approximation properties of networks that use random node features as input (Abboud et al., 2020).

Our comprehensive experimental study explores the effects of the propagation and normalization operators, and the number of steps in the RFP trajectory, on both node-level and graph-level tasks. The results indicate that RFP improves the performance of MPNNs, with or without spectral and random encoding as well as other recent methods.

Contributions. The contributions of this work are as follows: (1) We develop RFP — a novel PE scheme that combines the benefits of RNF and spectral PE, based on iterative propagation and normalization of random features; (2) We describe a method for learning the propagation operator rather than using a predefined one; (3) We present a theoretical analysis of our PE, giving insight into the design choices we have made ; and (4) We evaluate RFP in a series of node and graph classification tasks demonstrating its benefit.

2. Related Work

Random node features. Random initializations are widely used in stochastic approximation algorithms such as matrix trace estimation (Hutchinson, 1989) and subgraph counting (Avron, 2010). In the context of GNNs, random input features were used to improve the expressiveness of MPNNs (Dasoulas et al., 2019; Puny et al., 2020; Abboud et al., 2020; Sato et al., 2021). In particular, Puny et al. (2020); Abboud et al. (2020), proved a universal approximation theorem (with high probability) for GNNs that use random node features. While theoretically powerful, such a strategy did not yield consistently improved performance on real-world datasets (see Bevilacqua et al. (2022) and Section 5).

Spectral positional encoding. Embedding nodes using the eigendecomposition of graph operators is a classical technique in data analysis, see e.g., Belkin & Niyogi (2003); Coifman & Lafon (2006). As these embeddings contain valuable information, recent work has proposed using them as PE to overcome the limitations of MPNNs (Dwivedi et al., 2020) and as inputs to transformers when applied to graph data (Dwivedi & Bresson, 2021). Popular instances of these methods suggest using the graph Laplacian eigenvectors as initial node features (Dwivedi et al., 2020; Dwivedi & Bresson, 2021), but several improvements have been recently

proposed (Kreuzer et al., 2021; Wang et al., 2022; Lim et al., 2023; Rampasek et al., 2022; Maskey et al., 2022). These methods use eigenvectors as PE, while we demonstrate that using the trajectory towards computing these eigenvectors, including the initial RNF, yields better performance, and has favorable theoretical properties.

Successive application of propagation operators. The concept of utilizing node features that underwent several applications of graph operators has been proposed and studied in previous papers (Zhou et al., 2003; Wang & Leskovec, 2020). Differently from our procedure, the output of the propagation steps for these methods is solely determined by the input graph connectivity and partially known node labels, and involves no consideration of input or random node features. The combination of original input node features and propagation steps has been explored in Klicpera et al. (2018); Wu et al. (2019); Frasca et al. (2020); Huang et al. (2021). However, these works have different motivations, propose substantially different architectures, and do not make use of random node features. Another recent paper (Dwivedi et al., 2022) suggests using learnable layers in order to enhance positional encoding based on the powers of random walk matrices.

Closely related to our work is a very recent work by Huang et al. (2022) that adds a matrix inversion-based normalization step after each propagation step, resulting in a propagation-normalization procedure similar to ours. The authors show that when the node feature matrix has full rank, their iterative process converges to the dominant k eigenvectors of the used operator, where k is the number of initial node features, and present an accuracy improvement over previous approaches. Their approach, however, relies on the *original* raw input node features. In contrast, our approach advocates the use of RNF, different normalization schemes, and several algorithmic enhancements like learning the propagation operator. In Section 4 we prove why starting from random features is a better choice than the original node features in some cases. In Section 5 we show that our approach achieves better results in practice.

3. Method

The following section describes the proposed method. We begin with a brief overview of the approach before discussing its main components in more detail.

Notation. An undirected graph is defined by the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of n vertices and $\mathcal{E}$ is a set of m edges. Let us denote by $\mathbf{f}_i \in \mathbb{R}^c$ the feature vector of the i-th node of $\mathcal{G}$ with c channels. Also, we denote the adjacency matrix by $\mathbf{A}$, the diagonal degree matrix by $\mathbf{D}$, and the corresponding ones with added self-loops by $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$, respectively. The graph Laplacian (with self-loops considered) is given by $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$. Lastly, we denote the symmetrically normalized adjacency matrix by $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ and the symmetrically normalized graph Laplacian by $\hat{\mathbf{L}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{L}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$.

3.1. Overview of the framework

Our method is illustrated in Figure 1: we begin by sampling multiple vectors from a random distribution. Once the vectors have been generated, they are processed iteratively by alternating propagation and normalization steps. As a final step, we concatenate all the intermediate results of the process above and feed the result into a GNN as PE. The pseudocode of our method can be found in Appendix D.

Main components. At the core of our method, there are four main components. (1) The first component is the type of RNF used by our method. Two distributions are considered: Standard normal and Rademacher distributions[1]. Other distributions may also be suitable.

(2) Our second component is the propagation operator, which governs the information flow between nodes. A propagation operator is denoted by a matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ and can be either predefined or learned. (3) The third component of our method is the normalization function $\mathbf{N} : \mathbb{R}^{n \times k} \to \mathbb{R}^{n \times k}$, which returns a normalized representation of the current node features. The normalization may take the form of a simple ℓ_2 normalization of the propagated RNF or a more complex orthonormalization process.

(4) The fourth and last component is the architecture used to process the proposed PE. MPNNs or graph transformers are natural choices. In addition, we propose a method for processing multiple input trajectories using an equivariant DSS-GNN architecture (Bevilacqua et al., 2022) which incorporates set symmetries.

Detailed workflow. In order to compute our PE, we sample the random features $\mathbf{r} \in \mathbb{R}^{n \times k}$ from some joint probability distribution $\mathcal{D}$ and perform P of the following iterations:

$$\begin{aligned} \mathbf{a}^{(0)} &= \mathbf{r} \\ \hat{\mathbf{a}}^{(p)} &= \mathbf{S}\mathbf{a}^{(p-1)} \\ \mathbf{a}^{(p)} &= \begin{cases} \mathbf{N}(\hat{\mathbf{a}}^{(p)}), & \text{if } \mathrm{mod}(p, w) = 0 \\ \hat{\mathbf{a}}^{(p)}, & \text{otherwise} \end{cases} \end{aligned} . \tag{1}$$

Here, $\mathbf{S}$ is the propagation operator. $\mathbf{N}$ is the normalization function mentioned above, and w is a positive integer hyperparameter that controls the frequency of normalization steps.

We define a feature *trajectory* $\mathbf{t}$ as the concatenation of the initial random features $\mathbf{r}$ and the P steps defined in

[1] A Rademacher random variable has a probability of 0.5 to get the value 1 or -1.

Equation (1):

$$\mathbf{t} = \mathbf{r} \oplus \mathbf{a}^{(1)} \oplus \ldots \oplus \mathbf{a}^{(P)} \in \mathbb{R}^{n \times k(P+1)}, \tag{2}$$

where $\oplus$ denotes channel-wise concatenation. As discussed in detail in the previous work section, a similar propagation and normalization scheme, starting from the original node features, and using a different normalization function, was recently suggested in Huang et al. (2022).

In order to enrich our PE, it may be helpful to use *several* trajectory inputs that originate from different random samples $\mathbf{r}_i$. In that case, $\mathbf{t}$ represents the set of these trajectories:

$$\mathbf{t} = \{\mathbf{t}_1, \ldots, \mathbf{t}_B\}, \tag{3}$$

with $\mathbf{t}_b = \mathbf{r}_b \oplus \mathbf{a}_b^{(1)} \oplus \ldots \oplus \mathbf{a}_b^{(P)} \in \mathbb{R}^{n \times k(P+1)}$.

We concatenate and embed our PE $\mathbf{t}$, and input node features $\mathbf{f}^{\text{in}} \in \mathbb{R}^{n \times c^{\text{in}}}$, using a linear layer $\mathbf{K}_{\text{embed}}$ followed by an optional non-linear activation σ, as follows:

$$\mathbf{f}^{(0)} = \sigma(\mathbf{K}_{\text{embed}}(\mathbf{f}^{\text{in}} \oplus \mathbf{t})). \tag{4}$$

We regard $\mathbf{f}^{(0)}$ as the initial node embeddings obtained by our method. We then utilize several GNN layers to further process those initial node embeddings.

3.2. Propagation operators

The propagation operator $\mathbf{S}$ in (1) determines the flow of information between nodes in a graph and is represented as a matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$. The choice of this operator directly controls the resulting PE. We consider two types of propagation operators: (1) *Predefined* propagation operators based on the input graph connectivity, such as the adjacency matrix or graph Laplacian; and (2) *Learned* propagation operators that are generated based on the input graph using a parametric model. We now discuss these two types of operators.

Predefined propagation operators. We consider two popular predefined operators: the symmetrically normalized adjacency matrix with added self-loop $\hat{\mathbf{A}}$ (which is widely used in GNNs to propagate features, see e.g. Kipf & Welling (2017); Wu et al. (2019); Chen et al. (2020a)) and the analogous symmetrically normalized Laplacian matrix $\hat{\mathbf{L}}$. When used with RFP, we will see that these operators give access to the eigenvectors of the graph operator. Other predefined propagation operators can also be considered.

Learnable propagation operator.

We additionally propose to learn the propagation operator. The advantages of using learnable operators are twofold. First, propagation by fixed operators may not produce suitable features for the learning task at hand, so it may be beneficial to tune the propagation operator to the particular learning task. Second, our learnable operators can capture pairwise node interactions that are difficult to model using our previously proposed local operators. To learn the operator, we utilize several GNN layers applied to the original node features, followed by a multi-head self-attention mechanism (Vaswani et al., 2017) which computes a score for each pair of nodes in the graph (see Figure 2). While the predefined operators that we (and previous works) considered are local, the learnable operator, as we formulate it, is capable of capturing long-distance structural similarities. The reason for this is that the GNN layers before the attention mechanism encode the local neighborhood (both connectivity and node features) around each node, allowing the attention to measure the degree of similarity between those neighborhoods. Due to this, our learnable operator can easily connect two nodes that are very far apart in the graph but have similar local structures. Details about the learning procedure are in Appendix B.

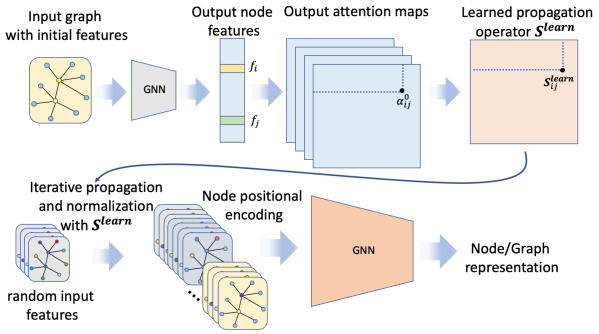

Figure 2. Workflow when using a learnable propagation operator. To obtain the propagation operator, we start from the original node features, process them with a GNN and apply an attention mechanism. We then use the resulting operator $\mathbf{S}^{\text{learn}}$ to generate our PE by applying it to random features.

3.3. Normalization function

A second key component of our RFP framework, is the normalization function. From a practical standpoint, a normalization step helps to control the magnitude of the features, which may grow exponentially with the number of iterations, depending on the propagation operator $\mathbf{S}$. Furthermore, as we discuss below, the choice of the normalization function has a significant effect on the feature trajectory $\mathbf{t}$ and its properties. In this paper, we consider two normalization functions: a simple channel-wise ℓ_2 normalization, and a more sophisticated orthonormalization described below.

ℓ_2 normalization. The first normalization we consider is a simple channel-wise ℓ_2 feature normalization, defined as:

$$\mathbf{N}^{\ell_2}(\hat{\mathbf{a}}^{(p)}) = \frac{\hat{\mathbf{a}}^{(p)}}{\|\hat{\mathbf{a}}^{(p)}\|_2}. \tag{5}$$

This normalization function is applied *independently* to each channel of the propagated node features, as described in Equation (1). Note that using this kind of normalization in tandem with the propagation steps, mimics the Power method iteration (Saad, 2011), a simple and popular algorithm for computing the dominant eigenvalue and eigenvector of a given diagonalizable matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$. The connection between our propagation and normalization schemes to the power iteration method sheds light on the feature trajectory we use as PE: given a diagonalizable propagation operator $\mathbf{S}$ with some random node features $\mathbf{r} \in \mathbb{R}^{n \times k}$, when $w = 1$, the p-th iteration described in Equation (1) is exactly the p-th power method iteration. This implies that in the limit, i.e., $P \to \infty$, under mild assumptions[2], all the channels, i.e., columns of $\mathbf{a}^{(P)} \in \mathbb{R}^{n \times k}$, will converge (up to sign) to the leading eigenvector of $\mathbf{S}$.

Orthonormalization. The second normalization method we consider is the joint, channel-wise orthonormalization of the node features. Given node features $\hat{\mathbf{a}}^{(p)} \in \mathbb{R}^{n \times k}$ at some trajectory iteration p, the proposed normalizing function reads:

$$\mathbf{N}^{\text{QR}}(\hat{\mathbf{a}}^{(p)}) = \mathbf{a}^{(p)} \quad , \tag{6}$$

where $\mathbf{a}^{(p)} \in \mathbb{R}^{n \times k}$ is such that ${\mathbf{a}^{(p)}}^\top \mathbf{a}^{(p)} = \mathbf{I}_k$ and the columns of $\mathbf{a}^{(p)}$ span the column space of $\hat{\mathbf{a}}^{(p)}$. In practice, several algorithms can realize Equation (6). We use PyTorch's built-in QR decomposition that implements the Householder orthogonalization scheme. Normalization according to Equation (6) offers a key difference compared to the ℓ_2 normalization from Equation (5): in the former, the iterations described in Equation (1) take the name of Subspace Iteration Method (Saad, 2011), a generalization of the power iteration method for computing *several* dominant eigenvectors simultaneously. In Section 4, we show that starting with RNF drawn from a continuous distribution, this process almost surely converges to the dominant k eigenvectors of the propagation operator $\mathbf{S}$.

3.4. Architectures

We propose to couple our RFP PE with two kinds of architectures. The first is a standard MPNN or transformer that consumes the initial embedding features $\mathbf{f}^{(0)}$ from Equation (4), and then performs several message-passing/attention layers based on some backbone layer such as GCN (Kipf & Welling, 2017), GraphConv (Morris et al., 2019), GIN (Xu et al., 2019) or GraphGPS (Rampasek et al., 2022).

As we show in Section 4, there are benefits to processing several trajectories. For example, one advantage of using multiple trajectories is that they allow GNNs to easily implement a substructure counting algorithm. However, processing several trajectories with the GNNs described above is not trivial. One way would be to concatenate them along the feature dimension and apply a standard MPNN as mentioned above. Unfortunately, it is unclear in what order these trajectories should be concatenated: they represent propagations originating from independently sampled random vectors with no canonical order. As a more effective approach to the problem, we suggest incorporating set symmetries by utilizing the DSS-GNN architecture (Maron et al., 2020; Bevilacqua et al., 2022), an architecture that was designed to process unordered sets of graphs.

[2]e.g. if $\mathbf{S}$ has a simple spectrum and the initialization has a non-zero projection on the dominant eigenvector.

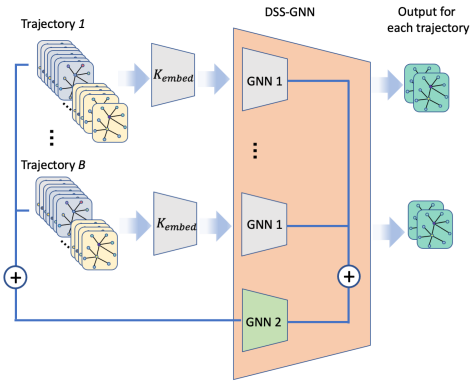

Figure 3. A single layer of our proposed DSS-GNN architecture for processing multiple trajectories in parallel. $\mathbf{K}_{\text{embed}}$ and GNN 1 are shared across all B trajectories.

Given multiple trajectories $\{\mathbf{t}_1, \ldots, \mathbf{t}_B\}$, $\mathbf{t}_b \in \mathbb{R}^{n \times k(P+1)}$, we first apply an embedding function $\mathbf{K}_{\text{embed}} : \mathbb{R}^{k(P+1)} \to \mathbb{R}^d$ to each trajectory $\mathbf{t}_b$ independently, and then reshape these embeddings into a $B \times n \times d$ tensor $\mathbf{F}$. These tensors can be viewed as a collection of B graphs containing node features based on the above B trajectories and with the same connectivity as the original graph. Thus, we can process them using the DSS-GNN architecture, which updates the representations of all the nodes for all B graphs simultaneously by using layers of the following form

$$L(\mathbf{F})_b = \text{GNN}_1(\mathbf{f}_b) + \text{GNN}_2\left(\sum_{b=1}^{B} \mathbf{f}_b\right).$$

$\text{GNN}_1, \text{GNN}_2$ are any GNNs, and $\mathbf{f}_b$ and $L(\mathbf{F})_b$ refer respectively to the given and the obtained features of the b-th graph in the set. Figure 3 illustrates the DSS-GNN layer. In graph-level tasks, after several such layers, a readout function is applied to each graph independently, followed by a Deepsets architecture (Zaheer et al., 2017) applied to the outputs. In node-level tasks, the B dimension in the output tensor is aggregated in a permutation-invariant manner.

4. Theoretical Analysis

In this section, we explain the main design choices taken in the development of our RFP framework. Specifically, we elaborate on the following points: (1) The importance of using multiple random initializations; (2) The significance of incorporating early propagation iterations in our RFP PE; and (3) The benefits of using random node features as a starting point for the trajectories over using the original node features.

All results are proved in Appendix C, which also includes an analysis of the time complexity of our approach.

4.1. Expressiveness

Triangle counting. The ability to count triangles is a fundamental skill of interest for a graph learning model. On the theoretical side, triangles are the simplest substructures standard MPNNs provably cannot count (Chen et al., 2020b); more practically, interesting triangular structures emerge in chemistry, e.g. cyclopropane and its derivatives, and social networks, e.g. locally clustered communities. Interestingly, and differently than standard GNNs, our approach is naturally suited for this kind of tasks: it can, in fact, implement the $\textsc{TraceTriangle}_R$ algorithm (Avron, 2010), a randomized algorithm for the estimation of triangle counts:

Proposition 4.1. *Let $\mathcal{G}_n$ be the class of all finite, simple, undirected graphs with n vertices. There exists a choice of hyperparameters $\mathcal{H}$, and weights $\mathcal{W}$, such that our algorithm (ref. Algorithm 1), equipped with either a DSS-GNN (Bevilacqua et al., 2022) or GraphConv (Morris et al., 2019) downstream architecture, exactly implements the* $\textsc{TraceTriangle}_R$ *algorithm by Avron (2010) over $\mathcal{G}_n$.*

These results may not appear surprising, in light of our RFP framework utilizing random node features, which are known to increase the expressiveness of MPNNs. However, the importance of Proposition 4.1 lies in showing how the most essential component of our approach, namely the *propagation* of random node features, unlocks specific computational abilities: as we detail in Corollary C.6 (Appendix C.1), our method can compute an approximation of triangle counting without resorting to expensive matrix multiplications.

Proposition 4.1 and its proof shed light on two other important aspects of our approach. The first one relates to the information enclosed in early propagation steps: $\textsc{TraceTriangle}_R$ can be implemented by considering only $\mathbf{a}^{(1)}, \mathbf{a}^{(2)}$, i.e., the features generated by the first two applications of the propagation operator. This suggests that meaningful patterns can already be extracted from initial propagation steps, before reaching eigenvector convergence. These early features, however, are not directly accessible by methods that employ, e.g. eigenvector-based PEs, as in Dwivedi et al. (2020). The second aspect is related to the fact that our method utilizes multiple trajectories originating from different random starting positions. While this characteristic is essential to implement $\textsc{TraceTriangle}_R$, more generally, it induces a natural alignment with randomized Monte Carlo algorithms and it may also contribute to enhanced performance in certain settings (see Section 5).

Other substructure features. Proposition 4.1 and Corollary C.6 can easily be extended so that RFP can approximate the computation of any $\text{trace}(\mathbf{S}^P)$, with $\mathbf{S}$ a generic (symmetric) operator and $P \in \mathbb{N}$ any integer power, opening up the possibility to extract more sophisticated structural features. For example, it is known that $\text{trace}(\mathbf{A}^P)$ counts the number of closed walks of length P and is related to the number of P-cycles c_P (Perepechko & Voropaev, 2012). A notable case corresponds to that of quadrangles, for which the closed form equation $c_4 = 1/8\big(\text{trace}(\mathbf{A}^4) + \text{trace}(\mathbf{A}^2) - 2\sum_v \deg^2(v)\big)$ is due to Harary & Manvel (1971).

Universal approximation. In addition to the results above, it is worth noting that our RFP framework can easily default to the RNI method introduced by Sato et al. (2021) and subsequently analyzed in its expressiveness by Abboud et al. (2020) and Puny et al. (2020). The latter work derives (ϵ, δ) universal approximation properties, which can be shown to be inherited by our RFP algorithm under the same assumptions. We formalize this result in Appendix C.2.

4.2. Comparison of random and initial input features

While our RFP focuses on the propagation and normalization steps of random node features, the same procedure can instead be applied to the initial, input node features, whenever those are available (Frasca et al., 2020; Huang et al., 2022). The convergence of the propagation-normalization iterations will then depend on the initial node feature matrix. In particular, in Huang et al. (2022), given initial node features $X \in \mathbb{R}^{n \times k}$, X must be of full rank in order to guarantee convergence to the k dominant eigenvectors of the propagation operator. This highlights two limitations caused by choosing the initial features as the starting point for the propagation-normalization iterations: (1) Convergence to the dominant eigenvectors is only guaranteed if the matrix of initial node features is of full rank, an assumption that might not always be met; (2) If convergence is reached, the number of dominant eigenvectors is exactly the number of initial input features, and thus cannot be customized to the task at hand. These limitations hinder the use of their method when learning graphs with no features or with low dimensional features, which often appear in graph classification tasks. On the contrary, the usage of random node features allows decoupling the number of eigenvectors to be obtained at convergence from the number of given initial features. We formalize that property in the following proposition, which we prove in Appendix C.3.

Proposition 4.2. *For any $k \leq n$, let $X = [X_1, \ldots, X_k] \in \mathbb{R}^{n \times k}$ be a concatenation of continuous i.i.d random variables, $X_i \in \mathbb{R}^n$ with probability density function f. Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be a graph operator, and denote by $\lambda_1, \ldots, \lambda_k$ the k dominant eigenvalues of $\mathbf{S}$ in decreasing order of magnitude. Assume $\mathbf{S}$ to be symmetric and that $|\lambda_i| > |\lambda_{i+1}|$, $1 \leq i \leq k$. Then, the Subspace Iteration algorithm, and hence the iteration of our RFP PE scheme, converges (up to sign) to the k-dominant eigenvectors of $\mathbf{S}$.*

5. Experiments

We conduct a comprehensive set of experiments to answer four main questions: (1) Does RFP outperform its natural baselines, namely RNF and spectral PE? (2) Is there a benefit to using one of our proposed architectures over the other? (3) How do different operators compare? Specifically, does the learnable operator improve performance? (4) How do different normalization schemes compare? Full details on experiments as well as additional results are in Appendix A.

Baselines. RFP is compared with a number of popular and recent methods. Specifically, we focus on the comparison of RNF (Abboud et al., 2020), and Laplacian PE (Dwivedi et al., 2020). When using Laplacian eigenvectors only as a baseline (denoted by $\hat{\mathbf{L}}$ EIGVECS†) we use the eigenvectors corresponding to the smallest eigenvalues, as it is standard in the literature (Dwivedi et al., 2020). For other baselines that involve PE based on other operators, we use the eigenvectors corresponding to the largest eigenvalues, to be consistent with the power method and RFP and allow easy comparison.

5.1. Graph-level tasks

Real World Data. We consider both the OGBG-MOLHIV (Hu* et al., 2020) and the ZINC-12k (500k budget) (Sterling & Irwin, 2015; Gómez-Bombarelli et al., 2018; Dwivedi et al., 2020) molecular datasets, where the tasks consist of predicting whether an input molecule inhibits HIV replication in the former, and the value of the constrained solubility in the latter. As a natural baseline, here, we also consider RWPE (Dwivedi et al., 2022) that utilizes powers of the random walk operator as positional encoding. In all baselines and RFP variants, we use GINE (Hu et al., 2020) as a backbone GNN, that extends GIN (Xu et al., 2019) by considering both node and edge features. Results are reported in Table 1. We observe that: (1) RFP significantly outperforms the natural baselines, such as Laplacian ($\hat{\mathbf{L}}$) and adjacency matrix ($\hat{\mathbf{A}}$) based PE, as well as RNF and their combination; (2) The RFP - DSS variant is consistently proven to be beneficial with respect to the GNN version; (3) Learning the propagation operator $\mathbf{S}^{\text{learn}}$ (which we couple with a DSS architecture) further yields better results. Overall, our RFP method can outperform a variety of methods, including provably expressive GNNs such as Corso et al. (2020); Beaini et al. (2021), as well as HIMP (Fey et al., 2020) and GSN (Bouritsas et al., 2022), that explicitly use domain-specific knowledge (and being outperformed only by CIN (Bodnar et al., 2021)). In Appendix A.4 we show that further improvements can be obtained by combining our RFP with domain-aware structural encoding methods.

Table 1. ZINC-12k and OGBG-MOLHIV datasets. Backbone is GINE for all baselines and RFP variants. † denotes eigenvectors corresponding to eigenvalues in ascending order of magnitude. The top three models are highlighted by **First**, **Second**, **Third**.

Method	ZINC-12K MAE ↓	OGBG-MOLHIV ROC-AUC(%) ↑
Standard GNNs		
GCN	0.321 ± 0.009	76.06 ± 0.97
GIN	0.163 ± 0.004	75.58 ± 1.40
PNA	0.133 ± 0.011	79.05 ± 1.32
DGN	–	79.70 ± 0.97
Domain-aware		
HIMP	–	78.80 ± 0.82
GSN	$\mathbf{0.101 \pm 0.010}$	$\mathbf{80.39 \pm 0.90}$
CIN	$\mathbf{0.079 \pm 0.006}$	$\mathbf{80.94 \pm 0.57}$
Natural baselines		
$\hat{\mathbf{L}}$ EIGVECS †	0.1557 ± 0.012	77.88 ± 1.82
$\hat{\mathbf{A}}$ EIGVECS	0.1501 ± 0.018	78.13 ± 1.61
$\hat{\mathbf{L}}, \hat{\mathbf{A}}$ EIGVECS	0.1434 ± 0.015	78.30 ± 1.47
RNF	0.1621 ± 0.014	75.98 ± 1.63
RNF & $\hat{\mathbf{L}}, \hat{\mathbf{A}}$ EIGVECS	0.1408 ± 0.020	78.96 ± 1.33
RWPE	0.1279 ± 0.005	78.62 ± 1.13
RFP (ours)		
RFP - ℓ_2 - $\hat{\mathbf{L}}, \hat{\mathbf{A}}$	0.1368 ± 0.010	77.91 ± 1.43
RFP - QR - $\hat{\mathbf{L}}, \hat{\mathbf{A}}$	0.1152 ± 0.006	79.83 ± 1.16
RFP - QR - $\hat{\mathbf{L}}, \hat{\mathbf{A}}, \mathbf{S}^{\text{learn}}$	0.1143 ± 0.008	79.94 ± 1.51
RFP - QR - $\hat{\mathbf{L}}, \hat{\mathbf{A}}$ -DSS	0.1117 ± 0.009	80.53 ± 1.04
RFP - QR - $\hat{\mathbf{L}}, \hat{\mathbf{A}}, \mathbf{S}^{\text{learn}}$ -DSS	$\mathbf{0.1106 \pm 0.012}$	$\mathbf{80.58 \pm 1.21}$

Synthetic Data. To validate our theoretical analysis on substructure counts, we conduct experiments on popular datasets (Chen et al., 2020b; Corso et al., 2020). Coherently with our theoretical claims, the results (deferred to Appendix A) show: (1) RFP attains larger expressiveness than standard MPNNs; (2) early propagation steps and the inclusion of multiple trajectories result in a lower test error.

5.2. Node-level tasks

To further highlight the benefits of RFP, we consider node-level tasks. We report results on 9 datasets consisting of both homophilic graphs (Cora (McCallum et al., 2000), Citeseer (Sen et al., 2008), and Pubmed (Namata et al., 2012)) and heterophilic graphs (Squirrel, Film, and Cham. (Rozemberczki et al., 2021), and Cornell, Texas, and Wisconsin (Pei et al., 2020)) [3]. In all datasets, we use the

[3]The definition of homophilic and heterophilic datasets can be found in Pei et al. (2020).

Table 2. Node classification accuracy (%) ↑. * denotes the best result out of several variants. An MLP backbone is used for Cornell, Texas, and Wisconsin, and a GCN backbone for the rest of the datasets. The top three models are highlighted by **First**, **Second**, **Third**.

Method Homophily	Squirrel 0.22	Film 0.22	Cham. 0.23	Corn. 0.30	Texas 0.11	Wisc. 0.21	Citeseer 0.74	Pubmed 0.80	Cora 0.81
Standard baselines									
MLP	28.77	36.41	46.24	80.89	80.81	83.88	74.16	87.05	75.59
GCN	23.96	26.86	28.18	52.70	52.16	48.92	73.68	88.13	85.77
Node classification designated architectures									
WRGAT	48.85	36.53	65.24	81.62	83.62	86.98	76.81	88.52	**88.20**
GGCN	**55.17**	**37.81**	**71.14**	**85.68**	**84.86**	86.86	**77.14**	**89.15**	87.95
H2GCN	36.48	35.70	60.11	82.70	**84.86**	**87.65**	**77.11**	**89.49**	87.87
GPRGNN	31.61	34.63	46.58	80.27	78.38	82.94	**77.13**	87.54	**87.95**
G^2-GCN	46.48	37.09	55.83	**86.49**	84.86	87.06	–	–	–
G^2-GraphSAGE	**64.26**	**37.14**	**71.40**	86.22	**87.57**	**87.84**	–	–	–
Natural baselines									
$\hat{\mathbf{L}}$ 64 EIGVECS	35.85	34.06	50.87	74.10	74.03	78.17	74.42	88.17	86.31
$\hat{\mathbf{A}}$ 64 EIGVECS	35.15	34.05	51.44	75.40	72.97	77.25	75.01	88.41	86.37
$\hat{\mathbf{A}}$ ALL EIGVECS	34.78	34.32	50.99	72.70	73.24	80.18	73.89	88.03	85.91
RNF	34.40	35.34	51.20	78.90	80.03	81.93	73.70	88.61	85.43
RNF & 64 $\hat{\mathbf{A}}$ EIGVECS	44.89	34.68	58.21	79.46	78.37	82.25	75.29	88.96	85.67
Propagation-based PE									
SGC	37.07	–	55.11	75.68	75.68	75.29	73.39	–	84.89
SIGN	40.97	–	60.11	76.76	75.14	78.43	73.27	–	83.92
PowerEmbed*	53.53	–	64.98	78.30	79.19	78.43	73.27	–	85.03
NFP - QR - $\hat{\mathbf{A}}$	46.85	27.72	61.25	65.40	78.23	62.97	73.89	88.86	85.93
RFP (ours)									
RFP - ℓ_2 - $\hat{\mathbf{A}}$	48.89	35.11	62.13	82.13	81.18	81.66	75.59	88.62	87.10
RFP - QR - $\hat{\mathbf{A}}$	54.28	36.87	65.98	85.13	84.74	86.08	76.48	**90.01**	87.99
RFP - QR - $\hat{\mathbf{A}}$ -DSS	54.78	36.93	66.16	85.69	85.10	86.35	76.92	90.00	**88.09**
RFP - QR - $\hat{\mathbf{L}}, \hat{\mathbf{A}}, \mathbf{S}^{\text{learn}}$ - DSS	**54.97**	**37.12**	**66.81**	**86.74**	**87.13**	**87.05**	76.53	89.96	87.99

standard 10 splits from Pei et al. (2020). The results are presented in Table 2, which reveals several interesting findings. First, our RFP - QR variants consistently improve the baselines GCN and MLP. Furthermore, it offers higher accuracy than the natural baselines RNF and Laplacian PE, in many cases by a large margin. Interestingly, we also find that using operator (e.g., $\hat{\mathbf{A}}$) eigenvectors as PE performs worse than the MLP baseline on Cornell, Texas, and Wisconsin datasets. RFP also consistently outperforms propagation-based PEs such as SGC (Wu et al., 2019), SIGN (Frasca et al., 2020), PowerEmbed (Huang et al., 2022), as well as Node Feature Propagation (NFP), a variant of our approach employing our propagation normalization mechanism to the original input node features rather than random node features. This observation shows that the choice of using random node features as a starting point to our trajectory is not only justified theoretically but also provides better results in practice. Lastly, we see that our RFP is in line with, or better than, recent methods that are node classification designated, such as H2GCN (Zhu et al., 2020), GGCN (Yan et al., 2021) and GPRGNN (Chien et al., 2021). On most datasets, our RFP also offers similar performance to the recent G^2 (Rusch et al., 2022); larger gaps are observed on Squirrel and Cham., where, however, their superior performance seems to be mostly driven by a different choice of the GNN layer (GraphSAGE). We note that our RFP can also be applied to large datasets, as opposed to the exact computation of the eigenvectors that is required in other PE methods. Thus, in Appendix A.3, we report the results obtained with our RFP on the large ogbn-arxiv dataset as well as the required computational resources.

5.3. Ablation study

We conduct an ablation study to investigate the effect of the number of propagation-normalization steps P as well as the number of chosen eigenvectors k. We report our results in Appendix A.4. In general, we found that increasing the number of steps and eigenvectors resulted in better scores.

5.4. Discussion

We have experimented with RFP on node-level and graph-level tasks. RFP consistently outperforms its natural base-

lines, in some cases by a wide margin. These results demonstrate the effectiveness of the main choices we made when designing our approach, including the fact that the full trajectory is informative and significant, and that starting the propagation from RNF is more effective than using initial features. Importantly, we also found orthonormalization ($\mathbf{N}^{\mathrm{QR}}$) to consistently outperform ℓ_2 normalization ($\mathbf{N}^{\ell_2}$) in all tasks.

In general, our method obtains competitive results on both node and graph tasks, while being outperformed only by domain-specific architectures (in the case of molecules) or architectures that are tailored for a specific task (in the case of node classification).

6. Conclusion

Our paper introduces RFP, a new family of PEs for graph learning, which combines two existing techniques, RNF and spectral PE, and has demonstrated favorable theoretical properties as well as better practical performance over these and even recent, task-specific techniques. We believe this work opens interesting future research directions, including exploring new operators and normalizations, designing novel parametric models for learning operators, and applying RFP for link prediction.

Acknowledgements

The research reported in this paper was supported by the Israeli Council for Higher Education (CHE) via the Data Science Research Center, Ben-Gurion University of the Negev, Israel. ME is supported by Kreitman High-tech scholarship.

References

Abboud, R., Ceylan, I. I., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

Avron, H. Counting triangles in large graphs using randomized matrix trace estimation. In *Workshop on Large-scale Data Mining: Theory and Applications*, volume 10, pp. 9, 2010.

Avron, H. and Toledo, S. Randomized algorithms for estimating the trace of an implicit symmetric positive semidefinite matrix. *Journal of the ACM*, 58(2):1 – 34, 4 2011.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W. L., Corso, G., and Liò, P. Directional graph networks. In *International Conference on Machine Learning*, 2021.

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 13–18 Jul 2020a.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 10383–10395. Curran Associates, Inc., 2020b.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

Coifman, R. R. and Lafon, S. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

Cybenko, G. V. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Dasoulas, G., Santos, L. D., Scaman, K., and Virmaux, A. Coloring graph neural networks for node disambiguation. *arXiv preprint arXiv:1912.06058*, 2019.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

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

Fey, M., Yuen, J.-G., and Weichert, F. Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workhop*, 2020.

Frasca, F., Rossi, E., Chamberlain, B., Eynard, D., Bronstein, M., and Monti, F. SIGN: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 7:15, 2020.

Frasca, F., Bevilacqua, B., Bronstein, M. M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. In *Advances in Neural Information Processing Systems*, 2022.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Jan 2018. ISSN 2374-7951. doi: 10.1021/acscentsci.7b00572.

Harary, F. and Manvel, B. On the number of cycles in a graph. *Matematický časopis*, 21(1):55 – 63, 1971.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Hu*, W., Liu*, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Huang, N. T., Villar, S., Priebe, C., Zheng, D., Huang, C., Yang, L., and Braverman, V. From local to global: Spectral-inspired graph neural networks. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

Huang, Q., He, H., Singh, A., Lim, S.-N., and Benson, A. Combining label propagation and simple models outperforms graph neural networks. In *International Conference on Learning Representations*, 2021.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Lim, D., Robinson, J. D., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. Sign and basis invariant networks for spectral graph representation learning. In *International Conference on Learning Representations*, 2023.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Maron, H., Litany, O., Chechik, G., and Fetaya, E. On learning sets of symmetric elements. In *International Conference on Machine Learning*, pp. 6734–6744. PMLR, 2020.

Maskey, S., Parviz, A., Thiessen, M., Stärk, H., Sadikaj, Y., and Maron, H. Generalized laplacian positional encoding for graph representation learning. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 2019.

Morris, C., Lipman, Y., Maron, H., Rieck, B., Kriege, N. M., Grohe, M., Fey, M., and Borgwardt, K. Weisfeiler and leman go machine learning: The story so far. *arXiv preprint arXiv:2112.09992*, 2021.

Namata, G., London, B., Getoor, L., Huang, B., and Edu, U. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, pp. 1, 2012.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

Perepechko, S. N. and Voropaev, A. N. The number of fixed length cycles in undirected graph. explicit formula in case of small lengths. *Discrete and Continuous Models and Applied Computational Science*, (2):6–12, 2012.

Puny, O., Ben-Hamu, H., and Lipman, Y. Global attention improves graph networks generalization. *arXiv preprint arXiv:2006.07846*, 2020.

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

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

Rusch, T. K., Chamberlain, B. P., Mahoney, M. W., Bronstein, M. M., and Mishra, S. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*, 2022.

Saad, Y. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 333–341. SIAM, 2021.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Sterling, T. and Irwin, J. J. ZINC 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 11 2015. doi: 10.1021/acs.jcim.5b00559.

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

Walker, A. J. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10:127–128, 4 1974. ISSN 0013-5194.

Wang, H. and Leskovec, J. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020.

Wang, H., Yin, H., Zhang, M., and Li, P. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*, 2022.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., and Koutra, D. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

Yun, C., Sra, S., and Jadbabaie, A. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. *Advances in Neural Information Processing Systems*, 32, 2019.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhang, M., Li, P., Xia, Y., Wang, K., and Jin, L. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

Zhou, D., Bousquet, O., Lal, T., Weston, J., and Schölkopf, B. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

Table 3. Graph Counting dataset MAE ↓. Backbone is GIN for all baselines and RFP variants. † denotes eigenvectors corresponding to eigenvalues in ascending order of magnitude. The top three models are highlighted by **First**, **Second**, **Third**.

Method	Triangle	Tailed-tri	Star	Cyc4
Standard models				
GCN	0.4186	0.3248	0.1798	0.2822
GIN	0.3569	0.2373	0.0224	0.2185
PNA	0.3532	0.2648	0.1278	0.2430
PPGN	**0.0089**	**0.0096**	0.0148	**0.0090**
Subgraph GNNs				
GNN-AK	0.0934	0.0751	0.0168	0.0726
GNN-AK-CTX	0.0885	0.0696	0.0162	0.0668
GNN-AK+	**0.0123**	**0.0112**	0.0150	**0.0126**
DSS-GNN (EGO+)	0.0450	0.0393	0.0129	0.0550
SUN (EGO+)	**0.0079**	**0.0080**	**0.0064**	**0.0105**
Natural baselines				
$\hat{\mathbf{L}}$ EIGVECS †	0.1681	0.1728	0.0168	0.1572
$\hat{\mathbf{A}}$ EIGVECS	0.1904	0.1714	0.0140	0.1533
$\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ EIGVECS	0.1694	0.1680	0.0121	0.1506
RNF	0.1775	0.1807	0.0189	0.1516
RNF & $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ EIGVECS	0.1512	0.1503	**0.0120**	0.1497
RFP (ours)				
RFP - ℓ_2 - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ - GINE	0.1521	0.1573	0.0177	0.1439
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ - GINE	0.0923	0.0826	0.0118	0.0991
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ - GINE-DSS	0.0820	0.0764	**0.0072**	0.0715

A. Experimental Results

A.1. Experimental settings

Our code is implemented using PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey & Lenssen, 2019), and all our experiments are run on Nvidia RTX3090 GPUs with 24GB of memory.

Natural baselines. RFP is compared with a number of popular and recent methods. In particular, we focus on the comparison of RNF (Abboud et al., 2020), Laplacian PE (Dwivedi et al., 2020) and PowerEmbed (Huang et al., 2022) which is arguably the closest approach to RFP. Additionally, we introduce further baselines based on our RFP method, such as including both the RFP and eigenvectors of the propagation operators without a complete RFP trajectory in order to demonstrate the importance of considering the complete trajectory. For the baseline case of Laplacian eigenvectors only (denoted by $\hat{\mathbf{L}}$ EIGVECS†) we use the eigenvectors corresponding to the smallest eigenvalues, in order to be consistent with the literature (Dwivedi et al., 2020). In the rest of the experiments that involve any other operator PE, we use the eigenvectors corresponding to the largest eigenvalues, to be consistent with the power method and our RFP.

Hyperparameters. We now list the hyperparameters in our experiments. We denote the learning rate by lr, weight decay by wd, and dropout probability by drop. The number of layers is denoted by L and the number of hidden channels by c. Besides those standard hyperparameters, our RFP requires two main hyperparameters: the number of propagations P, and the number of eigenvectors to be estimated k. In the case of DSS variants, we also choose the number of trajectories B. The hyperparameters values were determined using a grid search, and the considered values for those hyperparameters are reported in the following: $\mathrm{lr} \in \{1e-4, 1e-3, 1e-2, 1e-1\}$, $\mathrm{wd} \in \{0, 1e-6, 1e-5, 1e-4, 1e-3\}$, $\mathrm{drop} \in \{0, 0.5\}$, $L \in \{2, 4, 8, 16\}$, $c \in \{64, 128, 256\}$, $P \in \{2, 4, 8, 16, 32\}$, $k \in \{4, 8, 16, 32, 64\}$, $B \in \{5, 10\}$. Additional parameters, such as the number of epochs, the stopping criteria, and the evaluation procedure are as prescribed by the corresponding dataset (see e.g. Pei et al. (2020) and Frasca et al. (2022) for the node- and graph-level tasks, respectively). In the ZINC-12k dataset, we further make sure to remain within the prescribed 500k parameter budget. Finally, unless otherwise specified, we use a standard normal distribution to draw the initial random node features.

Table 4. Ablation study on the effect of the chosen number of eigenvectors k in RFP with fixed number of propagations $P = 16$ on ZINC-12k and OGBG-MOLHIV datasets.

Method	k	ZINC-12K MAE ↓	OGBG-MOLHIV ROC-AUC (%) ↑
GINE Backbone			
RFP - ℓ_2 - $\hat{\mathbf{L}}$, $\tilde{\mathbf{A}}$	64	0.1368 ± 0.010	77.91 ± 1.43
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	4	0.1305 ± 0.007	78.26 ± 1.51
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	8	0.1223 ± 0.009	79.43 ± 1.37
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	16	0.1152 ± 0.006	79.83 ± 1.16
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$, $\mathbf{S}^{\text{learn}}$	16	0.1143 ± 0.008	80.29 ± 1.01
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ - DSS	16	0.1117 ± 0.009	80.53 ± 1.04
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$, $\mathbf{S}^{\text{learn}}$ - DSS	16	0.1106 ± 0.012	80.58 ± 1.21
GraphGPS Backbone			
RFP - ℓ_2 - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	64	0.1322 ± 0.014	77.98 ± 1.57
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	16	0.1100 ± 0.005	79.95 ± 1.41
Combining RFP with domain-aware PE			
RFP+GSN - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	16	0.0734 ± 0.007	80.07 ± 1.19

Table 5. Ablation study on the effect of the chosen number of eigenvectors k in RFP with fixed number of propagations $P = 16$ on the Graph Counting dataset test MAE ↓. GIN is used as backbone.

Method	k	Triangle	Tailed-tri	Star	Cyc4
RFP_{ℓ_2}	16	0.1521	0.1573	0.0177	0.1439
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	4	0.1322	0.01289	0.0126	0.1401
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	8	0.1231	0.1143	0.0119	0.1170
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$	16	0.0923	0.0826	0.0118	0.0991
RFP - QR - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ - DSS	16	0.0820	0.0764	0.0072	0.0715

A.2. Additional experiments: the synthetic Graph Counting dataset

Besides real-world datasets which are presented in the main text, we also experiment with a synthetic dataset that is typically used for assessing the expressiveness of GNN (Chen et al., 2020b; Corso et al., 2020). The objective is to count the number of sub-structures present in a graph, such as triangles or cycles of length 4. To be aligned with our theoretical results in Section 4 and Appendix C, we use the Rademacher distribution to draw random node features to be used in our RFP. In our experiments, we found similar performance when using normally distributed random node features. Furthermore, we also used the Rademacher distribution for all the other baselines besides the RNF, where we consider a normal distribution to be consistent with RNI (Abboud et al., 2020). We follow the experimental settings proposed in Zhao et al. (2022), and we minimize the mean absolute error (MAE) of the prediction. As baseline methods we consider several GNNs such as GIN (Xu et al., 2019), PPGN (Maron et al., 2019), PNA (Corso et al., 2020), GCN (Kipf & Welling, 2017), GNN-AK (Zhao et al., 2022), SUN (Frasca et al., 2022). We report the obtained MAE in Table 3, where we see that our RFP approach improves on the natural baselines, such as RNF and PE. Also, note that here, PowerEmbed (Huang et al., 2022), which performs several propagation and normalization steps, cannot be directly compared with RFP because of their reliance on input node feature rather than random node features, which are completely absent in this dataset.

A.3. Node classification on a large graph

We evaluate RFP on the ogbg-arxiv dataset (Hu et al., 2020), which contains 169,343 nodes and 1,166,243 edges, and report the result in Table 7. It was straightforward to train RFP on that dataset using a GCN backbone with P=16 propagation steps and k=64 eigenvectors. A feed-forward pass takes 61.44 milliseconds on an RTX-3090 Nvidia GPU. We also note, that while using our RFP requires more computations, when not learning the propagation operator (as in the experiment reported in Table 7), our RFP does not incur significant memory overhead because it does not require performing backpropagation through the RFP layers, besides the initial embedding layer between the actual inputs node embedding and the RFP

Table 6. Node classification accuracy (%) ↑ on homophilic and heterophilic datasets.

Method	OP	k (vecs)	# props	Squirrel	Film	Cham.	Corn.	Texas	Wisc.	Cora	Cite	Pub
Homophily	–	–	–	0.22	0.22	0.23	0.30	0.11	0.21	0.81	0.74	0.80
EIGVEC-PE	$\hat{\mathbf{L}}$	64	–	35.85	34.06	50.87	74.10	74.03	78.17	86.31	74.42	88.17
EIGVEC-PE	$\hat{\mathbf{A}}$ (TopAbs)	64	–	35.15	34.05	51.44	75.40	72.97	77.25	86.37	75.01	88.41
EIGVEC-PE ALL	–	$\lvert\mathcal{V}\rvert$	–	34.78	34.32	50.99	72.70	73.24	80.18	85.91	73.89	88.03
RNF	–	64	0	34.40	35.34	51.20	78.90	80.03	81.93	85.43	73.70	88.61
RNF	–	$\lvert\mathcal{V}\rvert$	0	34.36	34.74	53.28	79.30	80.00	81.97	84.56	72.29	88.51
RNF & $\hat{\mathbf{A}}$ EIGVECS	$\hat{\mathbf{A}}$	64	–	44.89	34.68	58.21	79.46	78.37	82.25	85.67	75.29	88.96
NFP - QR	$\hat{\mathbf{A}}$	64	16	46.85	27.72	61.25	65.40	78.23	62.97	73.89	88.86	85.93
RFP - ℓ_2	$\hat{\mathbf{L}}$	1	16	48.89	35.11	62.13	82.13	81.18	81.66	87.10	75.59	88.62
RFP - QR	$\hat{\mathbf{A}}$	$\lvert\mathcal{V}\rvert$	16	54.23	36.61	64.77	85.07	85.41	86.27	85.68	75.16	89.00
RFP - QR	$\hat{\mathbf{A}}$	64	2	53.87	36.40	64.50	83.78	82.70	83.13	86.26	75.22	88.68
RFP - QR	$\hat{\mathbf{A}}$	64	4	54.11	36.44	65.16	84.59	83.78	84.90	86.67	75.37	88.76
RFP - QR	$\hat{\mathbf{A}}$	64	8	54.34	36.56	65.83	84.86	84.05	85.88	87.35	75.94	89.95
RFP - QR	$\hat{\mathbf{A}}$	64	16	54.28	36.87	65.98	85.13	84.32	86.08	87.99	76.40	90.01
RFP - QR	$\hat{\mathbf{A}}$	64	32	54.25	36.92	65.80	85.04	84.59	85.88	87.81	76.43	90.03
RFP - QR	$\hat{\mathbf{A}}$	4	16	51.96	35.88	64.01	83.24	82.70	82.16	86.93	76.11	89.66
RFP - QR	$\hat{\mathbf{A}}$	8	16	52.72	35.92	64.63	83.78	83.51	82.43	87.32	76.27	89.93
RFP - QR	$\hat{\mathbf{A}}$	16	16	53.87	36.51	65.11	84.59	83.78	82.97	87.65	76.39	90.00
RFP - QR	$\hat{\mathbf{A}}$	32	16	54.33	36.42	65.66	84.86	84.74	84.05	87.84	76.48	89.97
RFP - QR	$\hat{\mathbf{A}}$	64	16	54.28	36.87	65.98	85.13	84.32	86.08	87.99	76.40	90.01
RFP - QR	$\hat{\mathbf{A}}$	128	16	54.40	36.70	65.90	85.07	84.59	86.12	87.96	76.42	89.83
RFP - QR	$\hat{\mathbf{L}}, \hat{\mathbf{A}}, \mathbf{S}^{\text{learn}}$	64/32	16/8	54.56	36.95	66.34	86.40	86.72	86.88	87.92	76.41	89.63
RFP - QR - DSS	$\hat{\mathbf{A}}$	64/32	16/8	54.78	36.93	66.16	85.69	85.10	86.35	88.09	76.92	90.00
RFP - QR - DSS	$\hat{\mathbf{L}}, \hat{\mathbf{A}}, \mathbf{S}^{\text{learn}}$	64/32	16/8	54.97	37.12	66.81	86.74	87.13	87.05	87.99	76.53	89.96

Table 7. The obtained node classification accuracy(%) on ogbn-arxiv. The exact computation of the Laplacian $\hat{\mathbf{L}}$ eigenvectors required several hours and therefore the result is not available.

GCN	RNF + GCN	GCN + $\hat{\mathbf{L}}$ EIGVECS †	GCN + RFP (Ours)
71.08	70.91	N/A	71.97

embedded trajectory, as shown in Equation (4) in our paper. More precisely, a 2-layer GCN requires 7635 MB while our RFP combined with a 2-layer GCN backbone requires 9268 MB of memory.

From Table 7, we can see that:

(1) Adding plain random node features (RNF) to GCN on the large ogbg-arxiv dataset leads to slightly inferior results compared to the baseline of GCN.

(2) Computing the Laplacian eigenvectors as positional encoding (PE) is not practical for large graphs, because the positional encoding computation required several hours to complete on ogbg-arxiv.

(3) Combining GCN with our random feature propagation (RFP) mechanism improves the accuracy by 0.89%. We therefore deem that our RFP is also beneficial for large graphs for two reasons. First, it can be applied to large graphs, whereas using the exact Laplacian eigenvectors may not be practical, due to the computational overhead, and second, we found our RFP to empirically improve the baseline results.

A.4. Ablation study

Influence of the number of eigenvectors k. An important hyperparameter of our RFP is the number of eigenvectors to estimate, k. Table 4 shows the results on ZINC-12k and OGBG-MOLHIV datasets for different values of k. We further show the impact of k on the synthetic Graph Counting dataset in Table 5, and on the node classification datasets in Table 6. As can be seen in the tables, up to a typical threshold value of 16 and 64 on graph-level and node-level tasks, respectively, there is a significant benefit in the estimation of more eigenvectors. In the node classification datasets, which contain large

graphs, we also consider the case where all eigenvectors are estimated, where we do not see considerably better or worse performances.

Influence of the number of propagations P. We study the impact of the hyperparameter P, i.e., the number of propagation we take in our RFP. In Table 6 we show the node classification accuracy on several datasets for different values of P. We see that as a general rule of thumb, more propagations increase accuracy. This is in line with our interpretation of the RFP as the trajectory of generalized power methods. More specifically, more propagations are equivalent to more power method iterations, leading to more accurate eigenvector estimations of the considered graph operator $\hat{\mathbf{A}}$, $\hat{\mathbf{L}}$, or $\mathbf{S}^{\text{learn}}$. Also, we compare our RFP that propagates random features with a natural baseline of propagating the input node features, dubbed as NFP (node feature propagation). We see that our RFP offers superior results, further highlighting the importance of random node features.

Additional studies. We propose two additional settings to our RFP. (1) In addition to utilizing GINE as a backbone in our graph-level experiments, we also consider a combined transformer and GINE backbone, based on the GraphGPS architecture proposed by Rampasek et al. (2022). To understand the impact of the architectural change only, unlike in Rampasek et al. (2022), we do not use *structural* encodings, but rather only our RFP positional encodings. (2) We consider the combination of our RFP with a domain-aware structural encoding method, namely, GSN (Bouritsas et al., 2022). We report the obtained results on ZINC-12K and OGBG-MOLHIV, Table 4, where we can see the following: (1) compared to using GINE as a backbone only, we obtain slightly better results with GraphGPS. For example, using GraphGPS with RFP - ℓ_2 - $\hat{\mathbf{L}}$, $\hat{\mathbf{A}}$ reduces the test MAE on ZINC-12K from 0.1368 to 0.1322. (2) The improved results offered by combining RFP with GSN indicate that there is a synergy between the two methods. That is, the combination of both methods yields improved results that are closer to current state-of-the-art methods.

B. Learnable Propagation Operator

To implement our learnable propagation operator, we utilize a multi-head self-attention mechanism (Vaswani et al., 2017), denoted by MHA. The MHA computes h scores for each pair of nodes $(i, j) \in \mathcal{V} \times \mathcal{V}$. As an input to the MHA, we use a concatenation of the input node features $\mathbf{f}^{\text{in}}$ and the output of an MPNN applied to these input features, namely $\mathbf{f} = \mathbf{f}^{\text{in}} \oplus \mathrm{MPNN}(\mathbf{f}^{\text{in}})$. In practice, we found that using a single GCN (Kipf & Welling, 2017) layer as the MPNN component works well. We then obtain the pairwise attention scores by:

$$\mathrm{MHA}(\mathbf{f}_{\mathrm{i}}, \mathbf{f}_{\mathrm{j}}) = [\alpha^{1}_{\mathrm{ij}}, \ldots, \alpha^{\mathrm{H}}_{\mathrm{ij}}] \tag{7}$$

where $\mathbf{f}_i$ denotes the i-th row of $\mathbf{f}$. We then define the propagation operator as the average of these scores, i.e. $\mathbf{S}^{\text{learn}}_{ij} = \frac{1}{H}\sum_{h=1}^{H} \alpha^{h}_{ij}$. We then use the learnable operator $\mathbf{S}^{\text{learn}}$ in order to generate our trajectory. This is illustrated in Figure 2.

In our experiments, we found that the learnable operator may learn to encode different sparsity patterns (i.e., node interaction patterns), ranging from learning to choose a global node (that is connected to all of the nodes in the graph), to reducing neighborhood connectivity and enhancing self-loop weights. The former can be interpreted as learning to mimic the virtual global node approach, which is a well-known augmentation technique in GNNs (Bouritsas et al., 2022). Additionally, for fixed propagation operators, we have analyzed the information encoded in the trajectory in Section 4, and showed, that among other things, the trajectory to the eigenvectors provides us access to traces of powers of the propagation operators, which can be used to count relevant substructures such as cycles.

In the case of the learnable propagation operator, the trajectory provides at least two types of information:

(1) In a similar vein to the discussion in Section 4, the trajectory provides us information about the traces of powers of the learnable propagation operator. These traces can be interpreted as structural information that can be extracted from the new connectivity. It is possible, for example, to interpret the trace of the k-th power of the learnable operator as the probability that a random walk with k steps returns to the starting point. This provides structural information about a different, but still natural connectivity defined on the input graph compared to the original connectivity.

(2) Another possibly useful information that can be obtained from the trajectory of the learnable operator is the computation of features that describe the newly learnt connectivity and may provide useful hints for the downstream GNN. As shown Section 5, the learnable operator consistently improves or offers on-par performance compared with using RFP with pre-defined operators, mostly obtaining the former, that is, improving performance.

C. Theoretical Results

C.1. Triangle counting

It is well known that the number of triangles in a graph G with adjacency matrix $\mathbf{A}$ is computed as $c_3 = \frac{\text{Trace}(\mathbf{A}^3)}{6}$ (Harary & Manvel, 1971), that is, it is a sixth of the trace of the cubed adjacency matrix. As the trace of a matrix corresponds to the sum of its eigenvalues, the number of triangles can be estimated directly from an estimate of the cubed eigenvalues of the adjacency matrix $\mathbf{A}$. Our propagation scheme can, in principle, approximate these quantities by choosing operator $\mathbf{S} = \mathbf{A}^3$ and by letting the algorithm reach convergence. However, Avron (2010) argues that a significantly more efficient estimation of triangle counts in undirected graphs is constructed via Monte Carlo simulation, that is by running the following averaging operation:

$$\eta(G) = \frac{1}{M}\sum_{i=1}^{M} T_i \tag{8}$$

$$T_i = (\mathbf{y}_i^\top \mathbf{A}\mathbf{y}_i)/6 \tag{9}$$

$$\mathbf{y}_i = \mathbf{A}\mathbf{r}_i \tag{10}$$

where $\mathbf{r}_i$ is a $n \times 1$ vector with i.i.d. entries sampled from distribution $\mathcal{D}$. The author studies the impact of $\mathcal{D}$ on the error bound of the randomized algorithm, with particular emphasis on the cases where $\mathcal{D}$ is either a standard normal $\mathcal{N}(0,1)$ or a discrete Rademacher.

For the latter choice, as we report in Proposition 4.1, we show that our framework can exactly implement the above randomized algorithm and, therefore, inherit the approximation error bounds derived by Avron (2010).

Proof of Proposition 4.1. As for $\mathcal{H}$, let $\mathbf{S} = \mathbf{A}, P \geq 2, w \geq 3, B = M, k = 1$ ($P, w, B, k \in \mathbb{N}$) and $\mathbf{r} \in \{-1, +1\}^{n\times 1}$ have its entries sampled independently from a Rademacher distribution. Let $\mathbf{r}_i$ refer to the initial starting point for trajectory $\mathbf{t}_b, b = 1, \ldots, B$. In output from our propagation procedure we have the set of trajectories $\{\mathbf{t}_1, \ldots, \mathbf{t}_b, \ldots, \mathbf{t}_B\}$, where, for any b, $\mathbf{t}_b = \mathbf{a}_b^{(0)} \oplus \mathbf{a}_b^{(1)} \oplus \cdots \oplus \mathbf{a}_b^{(P)}$. These are gathered, along with initial node features, into the matrix $\mathbf{x}^{(0)} = (\bigoplus_{b=1}^{B} \mathbf{t}_i) \oplus \mathbf{x}$, obtained by stacking together the trajectories from the propagation phase, as well as the initial node features $\mathbf{x}$.

Importantly we note that, for any b, $\mathbf{u}_b = \mathbf{a}_b^{(0)} \oplus \mathbf{a}_b^{(1)} \oplus \mathbf{a}_b^{(2)} = \mathbf{r}_b \oplus \mathbf{A}\mathbf{r}_b \oplus \mathbf{A}^2\mathbf{r}_b$ in view of our specific choice of propagation operator and hyperparameter w. The concatenation of vectors $\mathbf{u}_b$'s, i.e. $\bigoplus_{b=1}^{B} \mathbf{u}_b$ will constitute the input of the downstream GNN architecture; the initial embedding layer $\mathbf{f}^{(0)}$ acts so extract these sub-trajectories from $\mathbf{x}^{(0)}$ (by discard all the other information originally contained therein). To this aim, it is sufficient to consider a linear embedding layer, that is, $\mathbf{f}^{(0)} \equiv \mathbf{K}_{\text{embed}}$. This last weight matrix is constructed as $\mathbf{K}_{\text{embed}} = K \oplus \mathbf{0}_{2B,d_{in}}$, with K a block diagonal matrix of dimension $2B \times BP$. Each of the B blocks in K is a $2 \times P$ matrix J whose only non-zero entries are those at first row and second column and second row and third column (which are one-valued). Effectively, each submatrix J has the role of 'selecting' only the values corresponding to the first and second propagation steps.

We now move to describe the downstream architecture. We start from the case where it is implemented as a (maximally expressive) message-passing architecture in the form proposed by Morris et al. (2019), that is as a stacking of T layers updating node representations as:

$$\mathbf{x}_v^{(t+1)} = \sigma\big(W_1^{(t)}\mathbf{x}_v^{(t)} + \sum_{w\sim v} W_2^{(t)}\mathbf{x}_w^{(t)} + b^{(t)}\big) \tag{11}$$

where σ is a ReLU non-linearity. The stacking is then followed by sum-readout aggregation operator and a final Multi Layer Perceptron (MLP) ψ computing a graph level output:

$$y = \psi\big(\sum_v \mathbf{x}_v^{(T)}\big) \tag{12}$$

The yet-to-describe architecture is able to start from $\bigoplus_{b=1}^{B} \mathbf{u}_b$, i.e. the output of $\mathbf{f}^{(0)}$, and complete the computation of the value of interest, i.e. $c_3(G) = \frac{1}{6M}\sum_{b=1}^{M}(\mathbf{y}_b^\top \mathbf{A}\mathbf{y}_b)$, with $\mathbf{y}_b = \mathbf{A}\mathbf{r}_b$. We rewrite term $\mathbf{y}_b^\top \mathbf{A}\mathbf{y}_b$ as: $\mathbf{y}_b^\top \cdot (\mathbf{A}\mathbf{y}_b) =$

$(\mathbf{A}\mathbf{r}_b) \cdot (\mathbf{A}\mathbf{A}\mathbf{r}_b) = (\mathbf{A}\mathbf{r}_b) \cdot (\mathbf{A}^2\mathbf{r}_b)$, that is, the dot product between, respectively, the second and third term of the b-th trajectory. Effectively, this product evaluates to $\sum_v (\mathbf{z}_b)_v$, $\mathbf{z}_b = (\mathbf{A}\mathbf{r}_b) \odot (\mathbf{A}^2\mathbf{r}_b)$, so that the expression is reformulated as: $c_3(G) = \frac{1}{6M}\sum_{b=1}^{M}(\mathbf{y}_b^\top \mathbf{A}\mathbf{y}_b) = \frac{1}{6M}\sum_{b=1}^{M}\sum_v (\mathbf{z}_b)_v$, $\mathbf{z}_b = (\mathbf{A}\mathbf{r}_b) \odot (\mathbf{A}^2\mathbf{r}_b)$.

We first show that terms $\mathbf{z}_b$'s can be computed by a stacking of layers of the form in Equation (11); then, that the remaining computation can be performed by the readout module described in Equation (12).

Although element-wise products are not natively supported by feed-forward neural networks, it would be possible to approximate them in view of the universal approximation theorem (Cybenko, 1989; Hornik et al., 1989). However, in our case, we require our model to only *memorize* the products between a finite number of operands, due to the discrete, finite nature of the Rademacher distribution that generates initial random features. In other words, we can exactly implement the calculation of $\mathbf{z}_b$'s thanks to the memorization theorem due to Yun et al. (2019):

Theorem C.1 (Theorem 3.1 from Yun et al. (2019))**.** *Consider any dataset $\{(x_i, y_i)\}_{i=1}^N$ such that all x_i's are distinct and all $y_i \in [-1, +1]^{d_y}$. Let $f_{\boldsymbol{\theta}}$ be a 3-layer ReLU-like MLP model, where d_1, d_2 refer to the dimensionality of its hidden representations. If $f_{\boldsymbol{\theta}}$ satisfies $4\lfloor d_1/4 \rfloor \lfloor d_2/(4d_y) \rfloor \geq N$, then there exist parameters $\boldsymbol{\theta}$ such that $y_i = f_{\boldsymbol{\theta}}(x_i)$ for all $i \in [N]$.*

The above theorem states the existence of a (small) ReLU-like network that can perfectly fit a given finite, well-formed dataset. ReLU-like networks are MLPs whose non-linearities are in the form $\sigma_R(x) = s_+ x$ for $x \geq 0$, $s_- x$ for $x < 0$. Family $\sigma_R(\cdot)$ includes the standard ReLU activation. Theorem C.1 can also be extended to the case where targets y_i's have their entries not necessarily constrained in the range $[-1, +1]$:

Proposition C.2 (Proposition 14 from Frasca et al. (2022))**.** *Consider any dataset $D = \{(x_i, y_i)\}_{i=1}^N$ such that all x_i's $\in \mathbb{R}^{d_x}$ are distinct and all $y_i \in \mathbb{R}^{d_y}$. There exists a 3-layer ReLU-like MLP $f^{(D)}$ such that $y_i = f^{(D)}(x_i)$ for all $i \in [N]$.*

The memorization theorems are, in fact, applicable in our context: this is fundamentally due to the fact that random inits $\mathbf{r}$'s come from a probability distribution with finite support, and the set of all possible adjacency matrices associated with n-nodes graphs is finite, so that there only exists a finite number of possible (2-step) propagation trajectories. In specific, consider the set of 2-dimensional vectors $\mathcal{X} = \left\{[(\mathbf{A}\mathbf{r})_i, (\mathbf{A}^2\mathbf{r})_i] \;\mid\; i \in \{1, \dots, n\}, \mathbf{A} \in \{0,1\}^{n\times n}, \mathbf{r} \in \{-1, +1\}^n\right\}$. This set is finite as its elements are vectors whose entries can only assume a finite set of values. This is because $[(\mathbf{A}\mathbf{r})_i, (\mathbf{A}^2\mathbf{r})_i] = [\mathbf{A}_{i,:}\mathbf{r}, \mathbf{A}^2_{i,:}\mathbf{r}]$, where $\mathbf{r}$ is from the finite $\{-1, +1\}^n$ and $\mathbf{A}_{i,:}, \mathbf{A}^2_{i,:}$ are rows from operators $\mathbf{A}, \mathbf{A}^2$ which are in a finite number due to $\mathbf{A}$ being in the finite $\{0,1\}^{n\times n}$. Now, let dataset D be constructed as $D = \left\{(x, y) \,|\, x = [x_a, x_b] \in \mathcal{X}, y = x_a \cdot x_b\right\}$. Dataset D 'lists' all possible pairwise multiplications that need to be calculated by our network in the computation of terms $(\mathbf{A}\mathbf{r}) \odot (\mathbf{A}^2\mathbf{r})$ above. D satisfies the hypotheses of Proposition C.2, which guarantees the existence of 3-layer ReLU-like MLP $f^{(D)}$ memorizing it.

Because of this, there also exists a 3-layer ReLU MLP f which, applied on the vectorized set of sub-trajectories $\bigoplus_{b=1}^B \mathbf{u}_b$, outputs the pairwise multiplications of the two channels in each of the $\mathbf{u}_b$'s, i.e. $\big((\mathbf{A}\mathbf{x}_1) \odot (\mathbf{A}^2\mathbf{x}_1)\big) \oplus \cdots \oplus \big((\mathbf{A}\mathbf{x}_B) \odot (\mathbf{A}^2\mathbf{x}_B)\big)$. Let $\left(W^{(i)}\right)_{i=1}^3$ be f's weight parameters; each weight matrix $W^{(i)}$ has a block diagonal structure with B blocks, where each block exactly corresponds to matrix $W^{(i,D)}$ in $f^{(D)}$. As for bias terms they are obtained concatenating, B times, the bias terms of $f^{(D)}$. The MLP f is trivially implemented by 3 message-passing layers in the form of Equation (11): it suffices to let $W_1^{(t)} = W^{(t)}, W_2^{(t)} = \mathbf{0}$, for $t = 1, 2, 3$ and the biases $b^{(t)}$ coincide the ones of f.

In output from these message-passing steps is $\bigoplus_{b=1}^B \mathbf{z}_b$. Here, we neglect the presence of the ReLU non-linearity in the last of these layers. This is legit assumption, as the effect of such activation can easily be nullified by properly choosing the weights in the preceding and subsequent linear transformations[4].

The computation of the dot product, for each sample b, is completed by the sum-readout operation in Equation (12), which effectively outputs vector $\bigoplus_{b=1}^B (6T_b)$. Lastly, ψ can easily and exactly implement the averaging operation $\frac{1}{6M}\sum_{b=1}^M 6T_b$, completing the computation of c_3: it only requires one dense layer with $1 \times B$ weight matrix $W_\psi = \frac{1}{6B}\vec{1}_B^\top$.

In the case of a DSS-GNN architecture (Bevilacqua et al., 2022), a bag of B subgraphs is formed. Subgraph b retains the same connectivity and initial node features of the input graph, but the latter ones are augmented with the trajectory resulting from the b-th random initialization. In other words, $A^b = A$, while $\mathbf{x}^{b,(0)} = \mathbf{t}_b \oplus \mathbf{x}$. Each $\mathbf{x}^{b,(0)}$ is embedded by the same

[4]Function $f(x) = R \cdot \sigma(E \cdot x)$, defined on $\mathbb{R}^d$, computes the identity function for $E = [\mathbf{I}_d | -\mathbf{I}_d]$, $R = \mathbf{I}_d \oplus \mathbf{I}_d$, σ being the ReLU non-linearity and $[\cdot|\cdot]$ representing vertical concatenation.

$\mathbf{f}^{(0)}$ layer, and then processed by DSS-GNN layers, which, equipped with maximally expressive graph convolutional layers in the form of Equation (11), read as:

$$\mathbf{x}_v^{k,(t+1)} = \sigma\big(W_{1,t}^1 \mathbf{x}_v^{k,(t)} + W_{2,t}^1 \sum_{w \sim v} \mathbf{x}_w^{k,(t)} + W_{1,t}^2 \sum_{b=1}^{B} \mathbf{x}_v^{b,(t)} + W_{2,t}^2 \sum_{w \sim v} \sum_{b=1}^{B} \mathbf{x}_w^{b,(t)}\big) \tag{13}$$

In DSS-GNN, a stacking of layers in the form of Equation (13), is followed by a subgraph readout operation:

$$\mathbf{x}^k = \rho\big(\sum_v \mathbf{x}_v^{k,(T)}\big) \tag{14}$$

and a final, global, readout layer:

$$y = \psi\big(\sum_{b=1}^{B} \mathbf{x}^b\big) \tag{15}$$

This architecture can implement the triangle counting algorithm. First, we consider the same $\mathbf{f}^{(0)}$ embedding layer we illustrated for the GNN architecture above. Then, a stacking of 3 DSS-GNN layers computes the element-wise product $(\mathbf{A}\mathbf{r}_b) \odot (\mathbf{A}^2\mathbf{r}_b)$, for each $b = 1, \dots, B$. Indeed, it is sufficient to let $W_{1,t}^1 = W^{(t,D)}$, all remaining weight matrices $W_{2,t}^1, W_{1,t}^2, W_{2,t}^2 = \mathbf{0}$, with $t = 1, 2, 3$ and all biases match those of $f^{(D)}$. Similar considerations w.r.t. the last ReLU activation as the ones above apply here.

The sum aggregation in Equation (14) completes the computation of the dot-product producing a scalar value for each subgraph. When ρ is a single linear layer with 1×1 weight matrix $W_\rho = [\frac{1}{6}]$, the subgraph readout operation on subgraph b outputs value T_b.

Finally, the readout operation in Equation (15) computes the averaging $\frac{1}{B}\sum_{b=1}^{B} T_b$, where, again ψ is a single linear layer with a 1×1 weight matrix $W_\psi = [\frac{1}{B}]$. □

An interesting aspect of the above proof is that the discussed constructions do not leverage any message-passing operation. Strictly speaking, the only architectural requirements to implement the desired dot products are: (i) a shared MLP applied to all node features in parallel, and (ii) a readout operation followed by a dense layer. Because of this, when starting from the same inputs, even a DeepSets model (Zaheer et al., 2017) would, in principle, be able to implement the TRACETRIANGLE$_R$ algorithm. This observation further confirms how meaningful structural patterns can easily be extracted from early propagation steps; in practice, downstream message-passing operations can then further refine this information into higher-level features.

As we will see next, the direct consequence of Proposition 4.1 is that, as for the TRACETRIANGLE$_R$ algorithm, our method can (ϵ, δ)-approximate the triangle counting function. This result is directly adapted from (Avron, 2010), and relies on the ability to implement the Hutchinson trace estimator:

Definition C.3. An *Hutchinson trace estimator* for a symmetric matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ is:

$$H_M(\mathbf{C}) = \frac{1}{M} \sum_{i=1}^{M} \mathbf{z}_i^\top \mathbf{C} \mathbf{z}_i \tag{16}$$

where $\mathbf{z}_i$'s are independent vectors in $\mathbb{R}^n$ whose entries are i.i.d. Rademacher random variables.

The Hutchinson estimator has been analyzed in (Avron & Toledo, 2011; Avron, 2010), where the authors prove the following

Lemma C.4 (Lemma 5 in (Avron, 2010))**.** *Let $\mathbf{C}$ a symmetric matrix with non-zero trace. For $M \geq 6\epsilon^{-2}\rho(\mathbf{C})^2 \ln(2\mathit{rank}(\mathbf{C})/\delta)$, the Hutchinson estimator $H_M(\mathbf{C})$ is an (ϵ, δ) approximator of $\mathit{trace}(\mathbf{C})$, that is:*

$$\mathit{Pr}\big(\ |H_M(\mathbf{C}) - \mathit{trace}(\mathbf{C})| \leq \epsilon \cdot \mathit{trace}(\mathbf{C})\ \big) \geq (1 - \delta) \tag{17}$$

where, for a symmetric matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$, $\rho(\mathbf{C}) = \frac{\sum_{i=1}^{n} |\lambda_i|}{\mathit{trace}(\mathbf{C})}$, and λ_i refers to the i-th eigenvalue of $\mathbf{C}$.

The following proposition relates the ability of our RFP algorithm to implement TRACETRIANGLE$_R$ and that of computing a specific Hutchinson estimator:

Proposition C.5. *Let $\mathbf{A}$ be the adjacency matrix associated with graph $G \in \mathcal{G}_n$. There exists a choice of hyperparameters $\mathcal{H}$, and weights $\mathcal{W}$, such that our RFP algorithm (ref. 1), equipped with either a DSS-GNN (Bevilacqua et al., 2022) or GraphConv (Morris et al., 2019) downstream architecture, computes the quantity $\frac{1}{6}H_b(\mathbf{A}^3)$, i.e., one sixth of the Hutchinson's estimator for trace$(\mathbf{A}^3)$.*

Proof. From Proposition 4.1 our approach can implement TRACETRIANGLE$_R$, that is, it outputs, for graph G, the value $\eta(G)$. At the same time, $\eta(G)$ can be rewritten as: $\eta(G) = \frac{1}{B}\sum_{b=1}^{B} T_b = \frac{1}{B}\sum_{b=1}^{B} \frac{\mathbf{y}_b^\top \mathbf{A}\mathbf{y}_b}{6} = \frac{1}{6}\frac{1}{B}\sum_{b=1}^{B} \mathbf{y}_b^\top \mathbf{A}\mathbf{y}_b = \frac{1}{6}\frac{1}{B}\sum_{b=1}^{B} (\mathbf{x}_b^\top \mathbf{A}^T)\mathbf{A}(\mathbf{A}\mathbf{x}_b)$. As $\mathbf{A}$ is symmetric, we have: $\eta(G) = \frac{1}{6}\frac{1}{B}\sum_{b=1}^{B} \mathbf{x}_b^\top(\mathbf{A}\mathbf{A}\mathbf{A})\mathbf{x}_b = \frac{1}{6}H_B(\mathbf{A}^3)$. □

From the above proposition we immediately derive the following

Corollary C.6. *Let $\delta > 0$ a failure probability and $\epsilon > 0$ a relative error. There exists a choice of hyperparameters $\mathcal{H}$ with $B \geq 6\epsilon^{-2}\rho(\mathbf{A}^3)^2 \ln(2rank(\mathbf{A}^3)/\delta)$ and a choice of weights $\mathcal{W}$ such that, for a graph $G \in \mathcal{G}_n$ associated with adjacency matrix $\mathbf{A}$, the output y_G of our RFP algorithm (ref. 1) is a (ϵ, δ) approximation of $c_3(G)$, i.e. the number of triangles of G:*

$$Pr\big(\ |y_G - c_3(G)| \leq \epsilon \cdot c_3(G)\ \big) \geq (1-\delta). \tag{18}$$

Proof of Corollary C.6. $\mathbf{A}^3$ is a symmetric matrix, since G is undirected and $\mathbf{A}$ is, thus, a symmetric matrix. From Proposition C.5, with $\mathbf{C} = \mathbf{A}^3$, with have:

$$\Pr\big(\ |H_B(\mathbf{A}^3) - \text{trace}(\mathbf{A}^3)| \leq \epsilon \cdot \text{trace}(\mathbf{A}^3)\ \big) \geq (1-\delta) \tag{19}$$

with $B \geq 6\epsilon^{-2}\rho(\mathbf{A}^3)^2 \ln(2\text{rank}(\mathbf{A}^3)/\delta)$. The event associated with expression (i) $|H_B(\mathbf{A}^3) - \text{trace}(\mathbf{A}^3)| \leq \epsilon \cdot \text{trace}(\mathbf{A}^3)$ is the same associated with expression (ii) $|y_G - c_3(G)| \leq \epsilon \cdot c_3(G)$, this proving the corollary. This is because we can rewrite (i) as $|6 \cdot y_G - \text{trace}(\mathbf{A}^3)| \leq \epsilon \cdot \text{trace}(\mathbf{A}^3)$, from which we obtain (ii) by multiplying both sides by the positive constant $1/6$: $|y_G - \frac{1}{6}\text{trace}(\mathbf{A}^3)| \leq \epsilon \cdot \frac{1}{6}\text{trace}(\mathbf{A}^3)$, and by recalling that $c_3(G) = \frac{\text{trace}(\mathbf{A}^3)}{6}$. □

C.2. Universal approximation

Abboud et al. (2020) and Puny et al. (2020) study the expressive power of the RNI model proposed by Sato et al. (2021). The model essentially applies a message-passing neural network on an input graph where node attributes are augmented with random features, acting as 'identifiers':

$$y_G = g\big(\mathbf{A}, \mathbf{f}^{\text{in}} \oplus \mathbf{r}\big) \tag{20}$$

where g is an MPNN and $\mathbf{f}^{\text{in}} \oplus \mathbf{r}$ is the horizontal concatenation of the initial node attribute matrix $\mathbf{x}$ and the random feature matrix $\mathbf{r}$. Despite slightly different technical assumptions and details, both the two works show that this architecture is an (ϵ, δ) universal approximator of functions on graphs. In this paper we take as reference the work by Puny et al. (2020), and build upon their result to derive Proposition C.7:

Proposition C.7. *Let Ω_n be a compact set of graphs on n vertices and f be a continuous target function defined thereon. Then, if random node features are sampled from a continuous, bounded distribution with zero mean and finite variance c, for all $\epsilon, \delta > 0$, then there exists a choice of hyperparameters $\mathcal{H}$, and weights $\mathcal{W}$, such that:*

$$\forall G \in \Omega_n : Pr\big(|f(G) - y_G| \leq \epsilon\big) \geq 1 - \delta \tag{21}$$

where y_G is the output of our algorithm on G for the considered choice of $\mathcal{H}, \mathcal{W}$.

Proof of Proposition C.7. The proposition immediately follows by showing a choice of hyperparameters and weights which makes our algorithm satisfy the premises of Puny et al. (2020, Proposition 1), which can then be invoked to guarantee the (ϵ, δ) universal approximation properties of our approach. As per our hypothesis, random features $\mathbf{r}$ are sampled from a continuous, bounded distribution; thus it is only left to show how algorithm can default to the RNI architecure in Equation (20). For any $P \geq 0$, it is sufficient to choose $B = 1$ to let the algorithm operate in a single trajectory regime, and construct module $\mathbf{f}^{(0)}$ in Equation (4) such that $\mathbf{f}^{(0)} = \sigma(\mathbf{K}_{\text{embed}}(\mathbf{f}^{\text{in}} \oplus \mathbf{t})) = \mathbf{f}^{\text{in}} \oplus \mathbf{r}$ (recall that the output of $\mathbf{f}^{(0)}$ is subsequently processed by a downstream MPNN). This construction of $\mathbf{f}^{(0)}$ is possible by properly choosing weights $\mathbf{K}_{\text{embed}}$ and by setting non-linearity σ to be the identity function. □

What would constitute a distribution $\mathcal{D}$ satisfying the hypotheses of Proposition C.7 (and (Puny et al., 2020, Proposition 1))? A valid choice is represented, for example, by the uniform distribution $\mathcal{U}_{[-1,1]}$, which is indeed zero-meaned, has variance $1/3$, is continuous, and bounded. The Rademacher distribution discussed above is, on the other hand, discrete. The reason why this may be problematic, from a theoretical standpoint, is that it can assign two nodes the same random features with non-zero probability. This may prevent the readout layer of the MPNN from unambiguously reconstructing the graph topology at hand, a step required in showing universal approximation. We remark that, in any case, the probability of a collision of initial Rademacher random features quickly decreases with the dimensionality of $\mathbf{r}$ and that, for Rademacher variables, specific expressiveness results can be proved nonetheless (see Appendix C.1).

C.3. Convergence to the operator eigenvectors

Lemma C.8. *Let $X_i \in \mathbb{R}^n$, $1 \leq i \leq k$ be continuous i.i.d. random variables with a continuous joint probability density function $f : \mathbb{R}^{n\times k} \to \mathbb{R}_+$. Denote their concatenation as $X = [X_1, \dots, X_k] \in \mathbb{R}^{n\times k}$. Let $\mathbf{Q} = [q_1, \dots, q_k] \in \mathbb{R}^{n\times k}$ be the matrix obtained by concatenating k arbitrary orthonormal vectors. Then Pr $\big($rank$(\mathbf{Q}\mathbf{Q}^\top X) = k\big) = 1$.*

Proof. We show that the complement event, namely $\{\text{rank}(\mathbf{Q}\mathbf{Q}^\top X) < k\}$ has zero probability. Let $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^\top X$, then $\text{rank}(\mathbf{Y}) < k$ if and only if $\text{rank}(\mathbf{Y}^\top\mathbf{Y}) < k$, which happens if and only if $\det(\mathbf{Y}^\top\mathbf{Y}) = 0$. Therefore we need to show that the probability of the event $\{\det(\mathbf{Y}^\top\mathbf{Y}) = 0\}$ is zero. Denote $g(\mathbf{Y}) = \det(\mathbf{Y}^\top\mathbf{Y})$, then g is a polynomial and we are interested in the probability of its zero-level set. But the zero-level set of a non-zero polynomial has zero Lebesgue measure. In our case, g is not the constant zero function since $g(\mathbf{Q}) = 1$. The result follows from the fact that $\Pr\left(g(\mathbf{Y}) = 0\right) = \int_{g^{-1}(\{0\})} f(x)dx = 0$, where the integral is zero since we integrate over a zero measure set. □

Proof of Proposition 4.2. We prove the proposition by showing we meet the assumptions of Saad (2011, Theorem 5.1). Although the theorem shows convergence to the Schur vectors $[q_1, \dots, q_k]$ associated with $\lambda_1, \dots, \lambda_k$, we note that for symmetric matrices they coincide with the k orthogonal eigenvectors.

We only need to show that $\text{rank}(P_i[X_1, \dots, X_i]) = i, 1 \leq i \leq k$ where P_i is the *spectral projector* associated with eigenvalues $\lambda_1, \dots, \lambda_i$. Since the eigenvalues are distinct by assumption, implying they are also *simple*, and the operator is symmetric (left and right eigenvectors coincide), the spectral projector can be written in terms of the corresponding eigenvectors as as $P_i = \sum_{j=1}^{i} q_j q_j^\top$, where we assume w.l.o.g. that q_j has unitary norm. Equivalently, we can write $P_i = \mathbf{Q}_i\mathbf{Q}_i^\top$ where $\mathbf{Q}_i = [q_1, \dots, q_i]$. Therefore we need to show that $\text{rank}(\mathbf{Q}_i\mathbf{Q}_i^\top[X_1, \dots, X_i]) = i, 1 \leq i \leq k$. This happens with probability 1 according to Lemma C.8. □

C.4. Computational complexity

If n, m refer to, respectively, the number of nodes and of edges of the input graph, the forward pass complexity of our method amounts to $T(n, m) = T_1(n, m) + T_2(n, m) + T_3(n, m)$, where T_1 is the complexity of sampling random node features, T_2 that of their propagation, and T_3 the complexity of the downstream GNN model. When $k \cdot B$ initial random features are sampled from a discrete distribution (such as a Rademacher one), then each node can be initialized in constant time via the Alias method (Walker, 1974), so that $T_1(n, m) = \mathcal{O}(kBn)$. T_2 accounts for two relevant operations, i.e. a normalization and a propagation, so that $T_2(n, m) = \mathcal{O}\big(P \cdot (k^2Bn + kBn^2)\big)$, where k^2Bn is the complexity of the QR normalization step (Saad, 2011) and kBn^2 that of the propagation one. This last can be reduced to kBm in the case of a fixed operator from a sparse graph. Lastly, the complexity of the downstream GNN is, generally, $C(n, m) = \mathcal{O}(L \cdot dm)$ for d-dimensional hidden representations and L convolutional layers; $\mathcal{O}(L \cdot dHn^2)$ in the case a Transformer architecture with H heads. When working on a large-scale, sparse graph such that parameters P, L, k, B, d can be considered small constants, the propagation and message-passing operations constitute the dominant factors in the computational complexity, which remains asymptotically linear in the number of edges.

D. Algorithm

In this section we present the pseudocode of our RFP framework. In particular, in Algorithm 1 we consider the case of a single trajectory (i.e. $B = 1$), coupled with a GNN architecture. Notably, the pseudocode can be easily extended to handle multiple trajectories $B > 1$, by simply repeating the propagation-normalization steps B times to obtain multiple trajectories $\{\mathbf{t}_1, \dots, \mathbf{t}_B\}$, $\mathbf{t}_b \in \mathbb{R}^{n\times k(P+1)}$. This set of trajectories can then be either concatenated and fed to the GNN, or processed with a DSS-GNN architecture, as explained in Section 3.4.

Algorithm 1 A feed-forward pass with a single RFP trajectory and a GNN backbone.

Input: Probability distribution $\mathcal{D}$, adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, input node features $\mathbf{f}^{\text{in}} \in \mathbb{R}^{n \times c^{\text{in}}}$, propagation operator (predefined or learnable) $\mathbf{S} \in \mathbb{R}^{n \times n}$, normalization function $\mathbf{N} : \mathbb{R}^{n \times k} \to \mathbb{R}^{n \times k}$, frequency of the normalization $w \geq 1$, number of propagation steps $P \geq 1$, a learnable embedding layer $\mathbf{K}_{\text{embed}}$ and a GNN model $\text{GNN}_{\mathcal{W}}$ with learnable parameters $\mathcal{W}$.

Initialize $\mathbf{r} \in \mathbb{R}^{n \times k}$ as $\mathbf{r} \sim \mathcal{D}$.
$\mathbf{a}^{(0)} = \mathbf{r}$
for $p = 1$ **to** P **do**
  $\hat{\mathbf{a}}^{(p)} = \mathbf{S}\mathbf{a}^{(p-1)}$
  if $\text{mod}(p, w) = 0$ **then**
    $\mathbf{a}^{(p)} = \mathbf{N}(\hat{\mathbf{a}}^{(p)})$
  else
    $\mathbf{a}^{(p)} = \hat{\mathbf{a}}^{(p)}$
  end if
end for
$\mathbf{t} = \mathbf{r} \oplus \mathbf{a}^{(1)} \oplus \ldots \oplus \mathbf{a}^{(P)}$
$\mathbf{f}^{(0)} = \sigma(\mathbf{K}_{\text{embed}}(\mathbf{f}^{\text{in}} \oplus \mathbf{t}))$
return $\text{GNN}_{\mathcal{W}}(\mathbf{f}^{(0)})$