Efficient Graph Field Integrators Meet Point Clouds

Krzysztof Choromanski [*1,2] **Arijit Sehanobish** [*3] **Han Lin** [*2] **Yunfan Zhao** [*2] **Eli Berger** [4] **Tetiana Parshakova** [5] **Alvin Pan** [2] **David Watkins** [6] **Tianyi Zhang** [2] **Valerii Likhosherstov** [7] **Somnath Basu Roy Chowdhury** [8] **Avinava Dubey** [1] **Deepali Jain** [1] **Tamas Sarlos** [1] **Snigdha Chaturvedi** [8] **Adrian Weller** [7,9]

Abstract

We present two new classes of algorithms for efficient field integration on graphs encoding point clouds. The first class, SeparatorFactorization (SF), leverages the bounded genus of point cloud mesh graphs, while the second class, RFDiffusion (RFD), uses popular ϵ-nearest-neighbor graph representations for point clouds. Both can be viewed as providing the functionality of Fast Multipole Methods (FMMs, Greengard & Rokhlin, 1987), which have had a tremendous impact on efficient integration, but for non-Euclidean spaces. We focus on geometries induced by distributions of walk lengths between points (e.g., shortest-path distance). We provide an extensive theoretical analysis of our algorithms, obtaining new results in structural graph theory as a byproduct. We also perform exhaustive empirical evaluation, including on-surface interpolation for rigid and deformable objects (particularly for mesh-dynamics modeling), Wasserstein distance computations for point clouds, and the Gromov-Wasserstein variant.

1. Introduction & Related Work

Let us consider a weighted undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E}, \mathrm{W})$, where: V stands for the set of vertices/nodes, E is the set of edges and $\mathrm{W} : \mathrm{E} \rightarrow \mathbb{R}_+$ encodes edge-weights. We assume that a tensor-field $\mathcal{F} : \mathrm{V} \rightarrow \mathbb{R}^{d_1 \times \ldots \times d_l}$ is defined on V, where: $d_1, \ldots, d_l$ stand for tensor dimensions. A kernel (similarity measure) $\mathrm{K} : \mathrm{V} \times \mathrm{V} \rightarrow \mathbb{R}$ on V is given. In this paper, we are interested in efficiently computing the expression $i(v)$ for each $v \in \mathrm{V}$, as defined below:

$$i(v) := \sum_{w \in \mathrm{V}} \mathrm{K}(w, v) \mathcal{F}(w). \tag{1}$$

The expression $i(v)$ can be interpreted as an integration of $\mathcal{F}$ on G with respect to measure $\mathrm{K}(\cdot, v)$. As such, it can also be thought of as a discrete approximation of the $\mathcal{F}$-field integration in the continuous non-Euclidean space, discretely approximated by G. We refer to the process of computing $i(v)$ for every $v \in \mathrm{V}$ as *graph-field integration* (GFI), see: Fig. 1. We write $N = |\mathrm{V}|$ for the size of V.

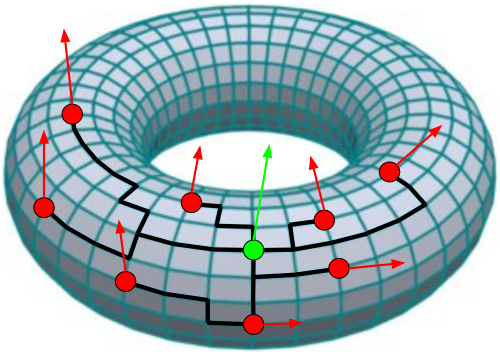

Figure 1. Visualization of the problem of integrating vector-field $\mathcal{F}$ on the mesh-graph. Vector $i(v)$ in the green-node v is defined as a weighted sum of the vectors $\mathcal{F}(w)$ in all the nodes (red arrows) with the coefficients $\mathrm{K}(w, v)$ given as $\mathrm{K}(w, v) = f(\mathrm{dist}(w, v))$ for a shortest-path-distance function dist between nodes of the mesh-graph and some function $f : \mathbb{R} \rightarrow \mathbb{R}$ (with $f(0) = 0$). The shortest-path-distance tree from v is marked in black.

A naive, brute-force approach to computing all $i(v)$ is to: **(1)** calculate a kernel matrix $\mathbf{K} = [\mathrm{K}(w, v)]_{w,v \in \mathrm{V}} \in \mathbb{R}^{N \times N}$ (***pre-processing***), **(2)** conduct $d_1 \cdot \ldots \cdot d_l$ matrix-vector multiplications: $\mathbf{K}\mathbf{v}^{c_1,\ldots,c_l}$ (***inference***) for $0 \leq c_i < d_i$, where $\mathbf{v}^{c_1,\ldots,c_l} \stackrel{\mathrm{def}}{=} \mathcal{F}(\cdot)[c_1] \ldots [c_l] \in \mathbb{R}^N$. Both steps are computationally expensive: inference requires $O(N^2 d_1 \cdot \ldots \cdot d_l)$ time; and pre-processing at least $O(N^2)$ (in practice, depending on kernel K, often at least $O(N^3)$). Therefore for large N, this approach becomes computationally infeasible.

It is thus natural to ask the following: *Can pre-processing or inference be performed in sub-quadratic time for the number of nodes of a graph?*

[*]Equal contribution [1]Google Research [2]Columbia University [3]Independent Researcher [4]Haifa University [5]Stanford University [6]The Boston Dynamics AI Institute [7]University of Cambridge [8]The University of North Carolina at Chapel Hill [9]The Alan Turing Institute. Correspondence to: Krzysztof Choromanski <kchoro@google.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

It is hopeless to provide a positive answer for an arbitrary graph G and kernel K; however, methods for certain subclasses have been extensively studied over decades. Probably the most prominent example is the family of *Fast Multipole Methods* (FMMs) (Greengard & Rokhlin, 1987; Möller et al., 2019; Gimbutas & Rokhlin, 2003; Yokota et al., 2016; Greengard et al., 2021; Cheng et al., 2006; Liska & Colonius, 2014; Takahashi et al., 2020). FMMs were originally developed for the N-body simulation problem (Ishiyama et al., 2022) in force-fields, where $\mathcal{F}$ might encode mass/charge distribution over points, K defines corresponding potential-field decay over distances and $i(v)$ calculates coordinates of the resultant forces in all N points. Since then, FMMs have been applied in a plethora of applications: (1) molecular/stellar dynamics, (2) interpolation with radial basis functions, and (3) solving differential equations: Poisson/Laplace (fluid dynamics, Barth, 1998), Maxwell's (electromagnetism, Darve & Have, 2004), Helmholtz (acoustic scattering problem, Gumerov & Duraiswami, 2021).

FMMs were developed for Euclidean spaces corresponding to grid graphs G embedded in $\mathbb{R}^d$ and the specific class of kernels K defined as functions of the dot-product similarity. However, several applications in machine learning involving point cloud and mesh-graph data require integrating more general graphs (defined on surfaces) approximating non-Euclidean metrics. Examples include: (a) computing Wasserstein distances between probabilistic distributions defined on meshes (Solomon et al., 2015), and (b) interpolation of the velocity fields given on meshes to model the complex dynamics of objects (Han et al., 2022).

In this paper, we present two new algorithms for the efficient (i.e., **sub-quadratic** in N) graph-field integration for graphs encoding point cloud data (where graph weights correspond to distances betweeen points). The first, $\mathrm{SeparatorFactorization}$ (SF), leverages bounded genus of point cloud mesh graphs, while the second, $\mathrm{RFDiffusion}$ (RFD), uses popular ϵ-nearest-neighbor (ϵ-NN) graph representations for point clouds. Both can be considered to provide the functionality of Fast Multipole Methods but in non-Euclidean spaces. We focus on geometries induced by distributions of walks' lengths between points (e.g., shortest-path distance). We provide an extensive theoretical analysis of the proposed algorithms and, as a byproduct, present new results in structural graph theory. We also perform exhaustive empirical evaluation, including on-surface interpolation for rigid and deformable objects (e.g., for mesh-dynamics modeling), Wasserstein distance computations for point clouds, including the Gromov-Wasserstein variant as well as point cloud classification. Our code is available at `https://github.com/topographers/efficient_graph_algorithms`.

To summarize, our main contributions are as follows:

1. We propose an $O(N\log^2(N))$ time complexity SF algorithm for approximate graph field integration on mesh-graphs, generalizing methods introduced by Choromanski et al. (2022) (Sec. 2.2, 2.3). The algorithm works for kernels of the form: $\mathrm{K}_f(w,v) = f(\mathrm{dist}(w,v))$ for the shortest-path-distance function dist and an **arbitrary** $f:\mathbb{R}\rightarrow\mathbb{R}$.

2. As a byproduct of methods developed for the SF algorithm, we construct **the first** efficient **exact** graph field integrator of $O(N\log^2(N))$ time complexity for unweighted graphs with bounded-length *geodesic-cycles* (e.g., cycles such that some shortest path between any two nodes of the cycle belongs to the cycle). For the important special case: $f_\lambda(x)=\exp(-\lambda x)$, we obtain additional computational gains resulting in $O(N\log^{1.383}(N))$ time complexity (Sec. 2.2).

3. We comprehensively compare the SF algorithm with alternative methods that approximate graph-induced metrics using the powerful technique of *low-distortion trees* (Bartal, 1996; Charikar et al., 1998; Fakcharoenphol et al., 2004; Abraham et al., 2008; Fellows et al., 2008; Bartal et al., 2022) (Sec. 3 and Appendix B).

4. We propose an $O(N)$ time complexity RFD algorithm for approximate graph field integration on generalized ϵ-NN graphs. By leveraging **r**andom-**f**eature-based embeddings, RFD decomposes ϵ-NN graphs into low-rank dot-product graph space (Li & Chang, 2014)(Sec. 2.4). RFD works for graph diffusion kernels of the form: $[\mathrm{K}(w,v)]_{w,v\in\mathrm{V}}=\exp(\lambda\mathbf{W}_\mathrm{G})$, where $\mathbf{W}_\mathrm{G}$ is the weighted adjacency matrix of G, and $\exp$ encodes matrix-exponentiation.

5. We comprehensively compare $\mathrm{RFDiffusion}$ with state-of-the-art algorithms for fast computation of the action of the exponentiated matrix (Al-Mohy & Higham, 2011; Musco et al., 2018).

The SF algorithm is a combinatorial method leveraging geometries defined by shortest path metrics in the form of the kernel $\mathrm{K}(w,v)=f(\mathrm{dist}(w,v))$. In contrast, the RFD algorithm is an algebraic approach utilizing geometries defined in terms of the distribution of walks of different lengths between the nodes, not only the shortest paths. The RFD approach complements the SF algorithm, acting on the ϵ-NN representation of the point cloud, which is a popular alternative to mesh-graphs used by the SF algorithm.

Random feature (RF) map methods are well-known to be an effective strategy to scale up kernel algorithms (Rahimi & Recht, 2007; 2008; Avron et al., 2017). This makes $\mathrm{RFDiffusion}$ the fastest method in our algorithmic portfolio, particularly well-suited for TPU/GPU-powered computations. However, it works for a specific (yet essential)

kernel, whereas the SF algorithm leverages a general class of shortest-path-induced kernels.

2. SeparatorFactorization and RFDiffusion

Defining geometries on graphs via walks. We start with the following class of kernels $\mathrm{K}^{\Lambda} : \mathrm{V} \times \mathrm{V} \rightarrow \mathbb{R}$ defined on the nodes of the graph G for the decreasing function $p : \mathbb{N} \rightarrow \mathbb{R}$ and a given hyper-parameter $\Lambda > 0$:

$$\mathrm{K}^{\Lambda}(w, v) = \sum_{k=0}^{\infty} p_{\Lambda}(k) n_k, \tag{2}$$

where n_k is the number of walks of length k between w and v. Taking $p_{\Lambda}(k) = \frac{\Lambda^k}{k!}$, one reconstructs a version of the so-called *graph diffusion kernel* (GDK). Using even a simpler formula: $p_{\Lambda}(k) = \Lambda^k = \exp(\log(\Lambda)k)$ we obtain another valid kernel related to the *Leontief Inverse matrix* (Bartolucci et al., 2020; Smola & Kondor, 2003). Note that the sum from the RHS of Eq. 2 starts de facto from $k = \mathrm{dist}(w, v)$ since $k \geq \mathrm{dist}(w, v)$. The first class of kernels considered in this paper is obtained by taking the latter formula for $p(k)$ and its leading p-coefficient from the sum in Eq. 2 (note that longer-walks are discounted as providing weaker ties between vertices). Instead of defining $\mathrm{K}(w, v) = \exp(-\lambda \mathrm{dist}(w, v))$ for $\lambda = -\log(\Lambda)$, we consider its generalized version for an **arbitrary** $f : \mathbb{R} \rightarrow \mathbb{R}$,

$$\mathrm{K}_f(w, v) = f(\mathrm{dist}(w, v)). \tag{3}$$

Thus the kernel becomes an arbitrary (potentially learnable) function of the shortest path distance. Such kernels are intensively studied for mesh modeling, where $\mathrm{dist}(\cdot, \cdot)$ approximates geodesic distances, see (Mory et al., 2009) and (Solomon et al., 2015). In the latter work, that kernel is ultimately replaced by a more computationally feasible variant of the diffusion kernel, see Sec. 3.2.

We also provide an efficient GFI mechanism in the setting where walks of all lengths are considered. Here we decide to work with the aforementioned GDK:

$$[\mathrm{K}^{\Lambda}_{\mathrm{GDK}}(w, v)]_{w,v \in \mathrm{V}} = \exp(\Lambda \cdot \mathrm{W}_{\mathrm{G}}), \tag{4}$$

for the weighted adjacency matrix W_{G} of G, where exp denotes matrix exponentiation.

2.1. Tractability and Bounded Genus Graphs

We start with the tractability concept recently introduced by Choromanski et al. (2022).

Definition 2.1 (tractable $(\mathcal{G}, f)$-pairs)**.** Let $\mathcal{G}$ be a family of weighted undirected graphs and let $f : \mathbb{R} \rightarrow \mathbb{C}$ be a function. Denote $\mathbf{K} = [\mathrm{K}_f(w, v)]_{w,v \in \mathrm{V}}$. We say that $(\mathcal{G}, f)$ is T-tractable if for any $\mathbf{x} \in \mathbb{R}^{|\mathrm{V}|}$, matrix-vector multiplication $\mathbf{K}\mathbf{x}$ can be computed in time $O(T)$. If $T = o(|\mathrm{V}|^2)$, then we say that $(\mathcal{G}, f)$ is tractable.

In Table 1, we summarize previously known results on T-tractable $(\mathcal{G}, f)$-pairs, all from (Choromanski et al., 2022).

Table 1. Summary of the known $(\mathcal{G}, f)$-tractability results (Choromanski et al., 2022). If not stated otherwise, considered graphs are unweighted. Below, $\mathrm{diam}(\mathrm{G})$ stands for the diameter of G.

$\mathcal{G}$	$f(z)$	T
weighted trees	$\exp(az + b)$ for given $a, b \in \mathbb{C}$	$\lvert\mathrm{V}\rvert$
unweighted trees	arbitrary	$\lvert\mathrm{V}\rvert \log^2(\lvert\mathrm{V}\rvert)$
unweighted trees	arbitrary	$\lvert\mathrm{V}\rvert \mathrm{diam}(\mathrm{G})$
d-dimensional grids	arbitrary	$\lvert\mathrm{V}\rvert \log(\lvert\mathrm{V}\rvert)$

We immediately realize that $(\mathcal{G}, f)$-tractability implies efficient GFI for kernels K_f. The results for trees from Table 1 can be elegantly extended to trigonometric functions f (still on trees) by using complex field $\mathbb{C}$, see Appendix, Sec. A.1.

In this section, we target mesh-graph representations of point clouds that are not trees. But they are not completely random. If the surface where the mesh graph lives does not have too many "holes", the mesh graph is very structured. This property can be precisely quantified as a *bounded genus*. The *genus* of a surface is the topologically invariant property defined as the largest number of non-intersecting simple closed curves that can be drawn on the surface without separating it (Lozano-Durán & Borrell, 2016).

Theorem 2.2 (Gilbert et al., 1984)**.** *The set of vertices* V *of graphs of genus* $\leq g$ *(i.e., embeddable with no edge-crossings on the surface of genus* g*) can be efficiently (in time* $O(|\mathrm{V}| + g)$*) partitioned into three subsets:* V_1, V_2 *and* $\mathcal{S}$ *such that:* $|\mathrm{V}_1|, |\mathrm{V}_2| \geq \frac{|\mathrm{V}|}{3}$, $|S| = O(\sqrt{(g+1)|\mathrm{V}|})$ *and furthermore there are no edges between* V_1 *and* V_2 *.*

We call set $\mathcal{S}$ a *balanced separator* since it splits the vertices of V into two "large" subsets (each of size $\geq c|\mathrm{V}|$ for some universal constant c; in our case $c = \frac{1}{3}$). Balanced separators are useful since they often enable using divide-and-conquer strategies to solve problems on graphs. As we show soon, this holds for the GFI problem.

2.2. Towards SeparatorFactorization: BCTW Graphs

Let us consider first an extreme case where bounded-size balanced separators exist. A prominent class of graphs with this property is a family of *bounded connected treewidth* (BCTW) graphs. We next introduce the concept of *treewidth* (tw), one of the most important graph parameters of modern structural graph theory.

Definition 2.3 (tree-decomposition & treewidth)**.** A tree-decomposition of a given undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is a tree T with nodes corresponding to subsets (bags) $X_1, \ldots, X_L$ of V satisfying the following:

- $\bigcup_{i=1}^{L} X_i = \mathrm{V}$,
- for every edge $\{u, w\} \in E$ there exists a bag X_i such that $u, w \in X_i$,

- for any two X_i, X_j, the subset $X_i \cap X_j$ is contained in all nodes on the (unique) path from X_i to X_j.

The *treewidth* of G is the minimum over different tree-decompositions of G of $\max_{i=1,\ldots,L} |X_i| - 1$.

We say that a family $\mathcal{G}$ of undirected and unweighted graphs has *bounded connected treewidth*, if each $\mathrm{G} \in \mathcal{G}$ has a tree-decomposition, where all the bags are connected graphs of bounded size. BCTW-graphs are extensions of trees. Every tree is a BCTW graph, but BCTW graphs can contain cycles (while trees cannot). It turns out that bags from the tree decomposition are themselves separators.

Next, we show that sparse BCTW graphs admit fast GFI:

Theorem 2.4. *If $\mathcal{G}$ is a family of bounded connected treewidth sparse graphs (i.e., with $|\mathrm{E}| = O(|\mathrm{V}|)$) then $(\mathcal{G}, f)$ is $|\mathrm{V}| \log^2(|\mathrm{V}|)$-tractable for any $f : \mathbb{R} \to \mathbb{C}$.*

Interestingly, a family $\mathcal{G}$ has bounded connected treewidth iff all the geodesic cycles of graphs $\mathrm{G} \in \mathcal{G}$ have bounded length (see Diestel & Müller, 2012). Thus, as a corollary, we immediately get the following result:

Corollary 2.5. *If $\mathcal{G}$ is a family of sparse graphs with geodesic cycles of bounded length, then $(\mathcal{G}, f)$ is $|\mathrm{V}| \log^2(|\mathrm{V}|)$-tractable for any function $f : \mathbb{R} \to \mathbb{C}$.*

Below we provide a sketch of the proof of Theorem 2.4, which also serves as pseudocode with a detailed explanation of each step. The full proof is given in Appendix Sec. A.2. This sketch will be sufficient to develop a "practical" version of the method that can be applied to bounded genus mesh graphs. We introduce extra notation for $\mathcal{Y}, \mathcal{Z} \subseteq \mathrm{V}$:

$$i^{\mathrm{G}}_{\mathcal{Z}}(v) \stackrel{\mathrm{def}}{=} \sum_{w \in \mathcal{Z}} \mathrm{K}(w, v)\mathcal{F}(w), \qquad i^{\mathrm{G}}_{\mathcal{Z}}(\mathcal{Y}) \stackrel{\mathrm{def}}{=} \{i^{\mathrm{G}}_{\mathcal{Z}}(y) : y \in \mathcal{Y}\}. \tag{5}$$

Proof sketch of Theorem 2.4:

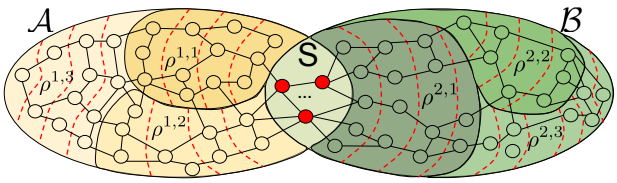

Figure 2. Visualization of the proof-sketch of Theorem 2.4. Subsets $\mathcal{A}$ and $\mathcal{B}$ are sliced based on the signature vectors $\rho^{x,y}$ (different shadows of yellow and green). Slices are further partitioned based on the distance from the separator $\mathcal{S}$ (dotted red lines).

Step 1: Balanced separation & initial integration. We start by finding a small (constant size) balanced separator $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ splitting V into two "large" subsets: $\mathcal{A}$ and $\mathcal{B}$ ($|\mathcal{A}|, |\mathcal{B}| > cN$ for some universal constant $c > 0$). It turns out that this can be done in time $O(N)$ (see: Sec. A.2). We compute $i^{\mathrm{G}}_{\mathrm{V}}(\mathcal{S})$ using Dijkstra's algorithm ($O(N \log(N))$ time complexity) or its improved variant ($O(N \log\log(N))$ time complexity; Thorup, 2003).

It suffices to compute $i^{\mathrm{G}}_{\mathrm{V}}(\mathcal{A} \cup \mathcal{B})$. Note that for every v:

$$i(v) = i^{\mathrm{G}}_{\mathcal{A}}(v) + i^{\mathrm{G}}_{\mathcal{B}}(v) + i^{\mathrm{G}}_{\mathcal{S}}(v). \tag{6}$$

Step 2: Computing $i^{\mathrm{G}}_{\mathcal{S}}(\mathcal{A} \cup \mathcal{B})$. As before, this can be done in time $O(N \log(N))$ (or even $O(N \log\log(N))$).

Step 3: Computing $i^{\mathrm{G}}_{\mathcal{A}}(\mathcal{A})$ and $i^{\mathrm{G}}_{\mathcal{B}}(\mathcal{B})$. This can be done by running the algorithm recursively on the sub-graphs $\mathrm{G}[\mathcal{A}]$, $\mathrm{G}[\mathcal{B}]$ of G induced by $\mathcal{A}$ and $\mathcal{B}$ respectively (for a rigorous proof, we actually need to run it for extended versions of $\mathrm{G}[\mathcal{A}]$ and $\mathrm{G}[\mathcal{B}]$ since the shortest path between two vertices in $\mathcal{A}/\mathcal{B}$ can potentially use vertices $\notin \mathcal{A}/\mathcal{B}$; crucially, as we show in Sec. A.2, those extended versions are obtained by adding only a **constant** number of extra vertices; for the practical variant we apply the simplified version though).

Step 4: Computing $i^{\mathrm{G}}_{\mathcal{A}}(\mathcal{B})$ and $i^{\mathrm{G}}_{\mathcal{B}}(\mathcal{A})$. We will show how to compute the latter. The former can be calculated in a completely analogous way.

Substep 4.1: $\mathcal{A}, \mathcal{B}$-slicing based on signature vectors. For every vertex $v \in \mathcal{A} \cup \mathcal{B}$, we define $\chi_v \in \mathbb{R}^{|\mathcal{S}|}$ as $\chi_v[k] = \mathrm{dist}(v, s_k)$ for $k = 1, \ldots, |\mathcal{S}|$. Write: $\chi_v = \tau_v + \rho_v$, where $\tau_v[i] = \min_{k \in \mathcal{S}} \mathrm{dist}(v, k), \forall i$. We call vector $\rho_v \in \mathbb{R}^{|\mathcal{S}|}$ the *signature vector* (sg-vect). The critical observation is that not only does this vector have bounded dimensionality (since $\mathcal{S}$ is of constant size), but a bounded number of different possible values of different dimensions, i.e., for every $i = 1, \ldots, |\mathcal{S}|$:

$$0 \leq \rho_v[i] \leq |\mathcal{S}| - 1. \tag{7}$$

This is an immediate consequence of the fact that the separator is connected. This implies that there is only a finite (yet super-exponentially large in $|\mathcal{S}|$!) number of different signature vectors ρ_v. We partition $\mathcal{A}$ into subsets corresponding to different signature vectors, called: $\rho^{1,1}, \rho^{1,2}, \ldots$ and similarly, partition $\mathcal{B}$ into subsets corresponding to different signature vectors: $\rho^{2,1}, \rho^{2,2}, \ldots$ (see Fig. 2).

Substep 4.2: Partitioning slices. Fix a subset $A_{\rho^{1,l}} \subseteq \mathcal{A}$ corresponding to some $\rho^{1,l}$ and a subset $B_{\rho^{2,t}} \subseteq \mathcal{B}$ corresponding to some $\rho^{2,t}$. Note that for every $v \in A_{\rho^{1,l}}$ and $w \in B_{\rho^{2,t}}$ the following holds:

$$\mathrm{dist}(w, v) = \tau_w[1] + \tau_v[1] + \min_{k \in \mathcal{S}}(\rho^{2,t}[k] + \rho^{1,l}[k]). \tag{8}$$

Furthermore, the last element of the RHS above does not depend on w and v, and $\mathrm{dist}(w, v)$ depends only on the

distances of w and v from $\mathcal{S}$. We thus partition subsets $A_{\rho^{1,l}}$ and $B_{\rho^{2,t}}$ based on the value of $\tau_v[1]$ and $\tau_w[1]$ accordingly. To compute $i^{\mathrm{G}}_{\mathcal{B}^{2,t}}(A_{\rho^{1,l}})$, it is thus sufficient to compute a sequence: $\{i^{\mathrm{G}}_{\mathcal{B}^{2,t}}(v_i)\}$ for $i = 0, 1, \ldots$ and where v_i is an arbitrary vertex of $A_{\rho^{1,l}}$ with $\mathrm{dist}(v_i, \mathcal{S}) = i$. This can be done via a linear transformation encoded by a Hankel matrix $\mathbf{W}$ as in the proof of Lemma 6.1 from (Choromanski et al., 2022) using Fast Fourier Transform (FFT) in time $O(N\log(N))$. Furthermore, if $f_\lambda(x) = \exp(-\lambda x)$, multiplication with $\mathbf{W}$ can be done in time $O(N)$. Putting together computations for all (constant) number of pairs: $(A_{\rho^{1,l}}, B_{\rho^{2,t}})$, we conclude that $i^{\mathcal{G}}_{\mathcal{B}}(\mathcal{A})$ can be computed in time $O(N\log(N))$ (or even $O(N\log\log(N))$ for $f_\lambda(x) = \exp(-\lambda x)$). Solving the corresponding time complexity recursion, we obtain total **pre-processing time** $O(N\log(N))$ and **inference time** $O(N\log^2(N))$ (or even $O(N\log^{1.38}(N))$ for $f_\lambda(x) = \exp(-\lambda x)$). This completes the proof sketch.

2.3. SeparatorFactorization

The SeparatorFactorization method is a straightforward relaxation of the the algorithm presented above. The relaxation to make the approach practical (for approximate GFI) is based on the following pillars:

1. **Separator truncation.** Replacing small $\mathcal{S}$ from Theorem 2.2 (not necessarily of constant size) with its subsampled constant-size subset $\mathcal{S}'$ (the other vertices of $\mathcal{S}$ are distributed across $\mathcal{A}$ and $\mathcal{B}$ randomly). Balanced separation is computed via the algorithmic version of Theorem 2.2 (see Fig. 3 for an example).

2. **Clustering signature vectors.** Instead of partitioning sets $\mathcal{A}$ and $\mathcal{B}$ based on sg-vects (their number is finite yet super-exponentially large in $|\mathcal{S}'|$), the partitioning is based on their hashed versions. We use a constant number of hashes. Every hashing mechanism that can be computed in time $O(N\log(N))$ is acceptable, and thus LSH methods can be applied (Shrivastava & Li, 2014). We found that in practice, initial partitioning based on sg-vect (substep 4.1) can be avoided. Good quality approximate GFI can be obtained by only one-level partitioning of $\mathcal{A}$ and $\mathcal{B}$ (based on the distance from $\mathcal{S}'$).

As in Sec. 2.2, this approach leads to $O(N\log(N))$ **pre-processing time** and $O(N\log^2(N))$ **inference time**. Furthermore, for weighted graphs, all the distances are effectively quantized (meaning natural numbers can approximate them). Finally, we stop the recursive unroll when the subsets $\mathcal{A}$ and $\mathcal{B}$ are small enough (when brute-force matrix-vector multiplication for GFI is fast enough).

2.4. RFDiffusion

In contrast to SeparatorFactorization above, the RFDiffusion algorithm leverages an ϵ-NN (Nearest Neighbor) representation of point clouds (see Fig. 3). This representation is particularly convenient for the graph diffusion kernel from Eq. 4 used by RFDiffusion.

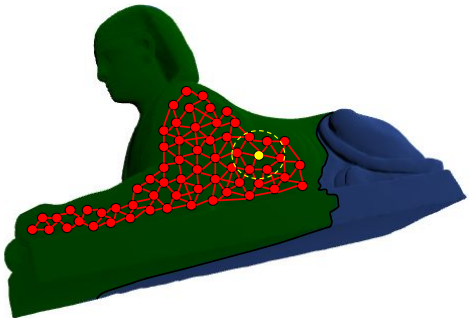

Figure 3. A sphinx mesh with $\mathbf{1.17M}$ faces, its first-level balanced separation obtained via SeparatorFactorization (with $\mathbf{685K}$ faces entirely in one class and $\mathbf{486K}$ entirely in the second one) and a visualization of the ϵ-NN graph leveraged by RFDiffusion. The graph is different from the mesh graph; particularly in this picture, it is not planar even though the mesh graph is.

RFDiffusion starts by producing a low-rank decomposition of the weighted adjacency matrix $\mathbf{W}_{\mathrm{G}}$ of G, defined via a set of vectors $\{n_i : i \in \mathrm{V}\} \subseteq \mathbb{R}^d$ and $f : \mathbb{R}^d \rightarrow \mathbb{R}$:

$$\mathbf{W}_{\mathrm{G}}(i,j) = f(\mathbf{n}_i - \mathbf{n}_j). \tag{9}$$

The *generalized ϵ-NN graphs* are special instantiations of such graphs, where f is defined as $f(\mathbf{z}) = h(\|\mathbf{z}\|)$ for non-increasing h with compact support, where vectors $\mathbf{n}_i$ are the points themselves and $\|\cdot\|$ is some norm (e.g. $f(\mathbf{z}) = \mathbb{1}[\|\mathbf{z}\| \leq \epsilon]$, as for the regular ϵ-NN graph).

Our goal is to rewrite: $\mathbf{W}_{\mathrm{G}}(i,j) \approx \phi(\mathbf{n}_i)^\top \psi(\mathbf{n}_j)$ for maps $\phi, \psi : \mathbb{R}^d \rightarrow \mathbb{C}^m$ and $m \ll N$. Note that (for $\mathbf{i}^2 = -1$):

$$\begin{aligned} f(\mathbf{z}) &= \int_{\mathbb{R}^d} \exp(2\pi\mathbf{i}\omega^\top \mathbf{z})\tau(\omega)d\omega \\ &= \int_{\mathbb{R}^d} \exp(2\pi\mathbf{i}\omega^\top \mathbf{z})\frac{\tau(\omega)}{p(\omega)}p(\omega)d\omega, \end{aligned} \tag{10}$$

where $\tau : \mathbb{R}^d \rightarrow \mathbb{C}$ is the Fourier Transform (FT) of f and p is a pdf function corresponding to some probability distribution $P \in \mathrm{Prob}(\mathbb{R}^d)$. Take $\omega_1, \ldots, \omega_m \overset{\mathrm{iid}}{\sim} P$. For $\mathbf{v} \in \mathbb{R}^d$, define $\rho_j = 2\pi\mathbf{i}\omega_j^\top \mathbf{v}$ and $\nu_i = \sqrt{\frac{\tau(\omega_i)}{p(\omega_i)}}$. Then, using Monte Carlo approximation, we can estimate: $\mathbf{W}_{\mathrm{G}}(i,j) \approx \phi(\mathbf{n}_i)^\top \psi(\mathbf{n}_j)$ for $\phi = \sigma_1$, $\psi = \sigma_{-1}$ and $\sigma_c(\mathbf{v}) = \frac{1}{\sqrt{m}}\left(\exp(2\pi c\mathbf{i}\omega_1^\top \mathbf{v})\nu_1, \ldots, \exp(2\pi c\mathbf{i}\omega_m^\top \mathbf{v})\nu_m\right)^\top$.

Note. Distribution P should 1) provide efficient sampling, 2) have easy-to-compute pdf, and 3) (ideally) provide low estimation variance. Here we use (truncated) Gaussian.

We conclude that we can decompose $\mathbf{W}_{\mathrm{G}}(i,j)$ as $\mathbf{W}_{\mathrm{G}}(i,j) = \mathbf{A}\mathbf{B}^T$, where the rows of $\mathbf{A} \in \mathbb{R}^{N\times m}$ and $\mathbf{B} \in \mathbb{R}^{N\times m}$ are given as: $\{\sigma_1(\mathbf{n}_j) : j \in \mathrm{V}\}$ and $\{\sigma_{-1}(\mathbf{n}_j) : j \in \mathrm{V}\}$ respectively. Now note that we have the following:

$$\begin{aligned}\exp(\Lambda \cdot \mathbf{A}\mathbf{B}^\top) &= \sum_{i=0}^{\infty} \frac{1}{i!}(\Lambda \mathbf{A}\mathbf{B}^\top)^i \\ &= \mathbf{I} + \sum_{i=0}^{\infty} \frac{1}{(i+1)!}\mathbf{A}(\Lambda \mathbf{B}^\top \mathbf{A})^{i+1}\mathbf{A}^{-1} \\ &= \mathbf{I} + \mathbf{A}[\exp(\Lambda \mathbf{B}^\top \mathbf{A}) - \mathbf{I}](\mathbf{B}^\top \mathbf{A})^{-1}\mathbf{B}^\top\end{aligned} \tag{11}$$

We can thus approximate: $\exp(\Lambda \cdot \mathbf{W}_{\mathrm{G}})\mathbf{x}$ for any vector $\mathbf{x} \in \mathbb{R}^N$ (which leads to the GFI algorithm) as:

$$\mathbf{x} + \mathbf{A}([\exp(\Lambda \mathbf{B}^\top \mathbf{A}) - \mathbf{I}]((\mathbf{B}^\top \mathbf{A})^{-1}(\mathbf{B}^\top \mathbf{x}))), \tag{12}$$

where brackets indicate the order of computations. We see that this algorithm has: (a) **pre-processing** time linear in N and cubic in the number of random features m, and (b) **inference** stage linear in N and quadratic in m. Algorithm RFDiffusion can be thought of as approximating the graph given by Eq. 9 via random feature map-based smoothing (in our applications with f given as a threshold function and $d = 3$) and has an excellent property - its running time is **independent** (Fig. 12 in Appendix E) of the number of edges of the graph (that is never explicitly materialized).

It remains to compute function τ. Fortunately, this is easy for several threshold functions f defining ϵ-NN graphs. For instance, for $f(\mathbf{z}) = \mathbb{1}[\|\mathbf{z}\|_1 \leq \epsilon]$ and $\xi \in \mathbb{R}^d$, we have:

$$\tau(\xi) = \prod_{i=1}^{d} \frac{\sin(2\epsilon\xi_i)}{\xi_i}, \tag{13}$$

and for $f(\mathbf{z}) = \mathbb{1}[\|\mathbf{z}\|_2 \leq \epsilon]$, τ is the d-th order *Bessel function* (Dattoli et al., 2004).

We quantify the quality of the estimation of the original ϵ-NN graph with L_1-norm via RFDiffusion (proof in Sec A.3). Analogous results can be derived from other norms.

Lemma 2.6. *Take the ϵ-NN point cloud graph in $\mathbb{R}^3$ with respect to the L_1-norm. For two given vertices v and w, denote by $\mathrm{MSE}(\widehat{\mathbf{W}}(v,w))$ the mean squared error of the* RFDiffusion*-based estimation of the true weight $\mathbf{W}(v,w)$ between v and w (defined as: 1 if $\mathrm{dist}_{\mathrm{L}_1}(v,w) \leq \epsilon$ and 0 otherwise). Let P be a Gaussian distribution truncated to the L_1-ball $\mathcal{B}(R)$ of radius R, used by RFD. Assume that v and w are encoded by $\mathbf{n}_v$ and $\mathbf{n}_w$. Then:*

$$\mathrm{MSE}(\widehat{\mathbf{W}}(v,w)) \leq \frac{1}{m}\left((2\pi)^{\frac{3}{2}} C(\Gamma_\epsilon(R))^d - \theta_1^2\right) + \theta_2, \tag{14}$$

for $\theta_1 = (f(\mathbf{z}) + \gamma)$, $\theta_2 = \gamma(2f(\mathbf{z}) + \gamma)$, $\mathbf{z} = \mathbf{n}_v - \mathbf{n}_w$, $\mathrm{C} = \int_{\mathcal{B}(R)} (2\pi)^{-\frac{3}{2}} \exp(-\frac{\|\mathbf{r}\|^2}{2}) d\mathbf{r}$, $\Gamma_\epsilon(R) = \int_{-R}^{R} \frac{\sin^2(\epsilon x)}{x^2} dx$ and $\gamma = -\int_{\mathbb{R}^d \setminus \mathcal{B}(R)} \cos(2\pi\omega^\top \mathbf{z}) \prod_{i=1}^{d} \frac{\sin(2\epsilon\omega_i)}{\omega_i} d\omega$.

3. Experiments

To evaluate SF and RFD, we choose two broad applications: a) interpolation on meshes and b) Wasserstein distances and barycenters computation on point clouds. The kernel matrices chosen for SF and RFD are $\mathrm{K}(i,j) := \exp(-\lambda \mathrm{dist}(i,j))$ and $\mathrm{K} := \exp(\lambda \mathbf{W}_{\mathrm{G}})$ respectively. Moreover we demonstrate the effectiveness of RFD kernel in various point cloud and graph classification tasks.

3.1. Interpolation on Meshes

In this section, we use our methods to predict the masked properties of meshes. In particular, we compare the computational efficiency of SF and RFD against baselines in predicting vertex normals and nodes' velocities in meshes.

Vertex normal prediction. In this setup, we predict the field of normals in vertices from its masked variant. We are given a set of nodes with vertex locations $\mathbf{x}_i \in \mathbb{R}^3$ and vertex normals $\mathbf{F}_i \in \mathbb{R}^3$ in a mesh G with vertex-set V. In each mesh, we randomly select a subset $\mathrm{V}' \subseteq \mathrm{V}$ with $|\mathrm{V}'| = 0.8|\mathrm{V}|$ and mask out their vertex normals (set as zero vectors). Our task is to predict the vertex normals of each masked node $i \in \mathrm{V}'$ computed as:

$$\mathbf{F}_i = \sum_{j \in \mathrm{V} \setminus \mathrm{V}'} \mathrm{K}(i,j)\mathbf{F}_j,$$

We perform a grid search on λ and other algorithm-specific hyper-parameters for each mesh. We report the result with the highest cosine similarity between predicted and ground truth vertex normals, averaged over all the nodes.

We run tests on **120 meshes** for 3D-printed objects with a wide range of sizes from the Thingi10k (Zhou & Jacobson, 2016) dataset (see Sec. C for details). Fig. 4 reports the pre-processing time, interpolation time, and cosine similarity for algorithms on meshes with different sizes $|\mathrm{V}|$. In the first row, we compare SF with brute-force (BF) (explicit kernel-matrix materialization followed by matrix-vector multiplications) and low-distortion tree-based algorithms such as Bartal trees (T-Bart-n; n is the number of trees, Bartal, 1996) and FRT trees (T-FRT, Fakcharoenphol et al., 2004) (see Appendix B for details). SF is the fastest in pre-processing and accurately interpolates on large meshes while BF, T-FRT, and T-Bart gradually run out of memory or time (OOM/OOT).

In the second row of Fig. 4, we compare RFD with three other algorithms for multiplications with matrix exponentials, including Bader's algorithm (Bader et al., 2019), Al-Mohy's algorithm (Al-Mohy & Higham, 2010), and Lanczos method (Orecchia et al., 2012). As the mesh size increases, the pre-processing time of Bader and Al-Mohy grows quickly. The performance of the Lanczos algorithm is positively correlated with hyper-parameter m, which controls the number of Arnoldi iterations. Even though we chose m to be relatively large (which affects the interpolation time), its performance still drops quickly as mesh size grows. In contrast, RFD scales well to large meshes. For example, on the mesh with **1.5M** nodes, RFD needs only **29.7** seconds for the pre-processing and **5.7** seconds for interpolation. Detailed ablation studies are given in Sec. E.1.

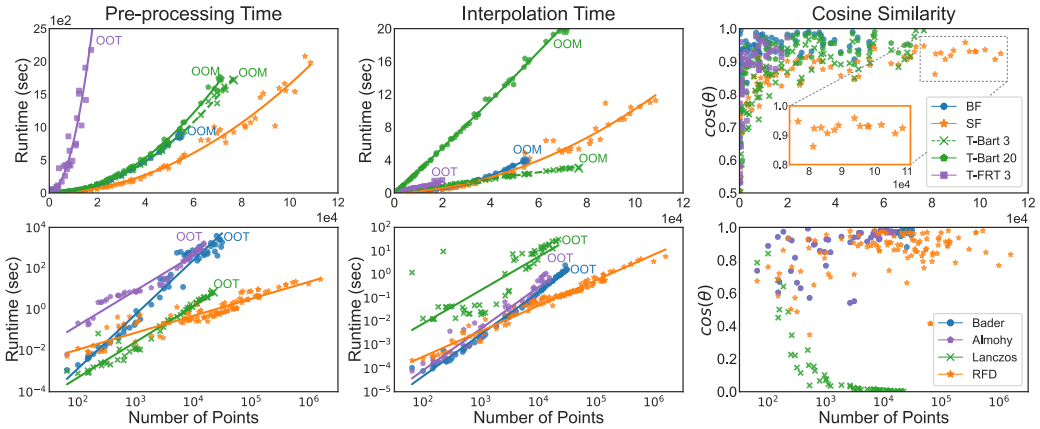

Figure 4. **First Row**: Vertex normal prediction with SF and comparison with relevant low-distortion trees methods. **Second Row**: The same task, but with RFD and the corresponding methods for multiplications with matrix exponentials. All methods except SF and RFD either went out of memory (OOM) or ran out of time (OOT). The two algorithms maintain high accuracy even on large meshes.

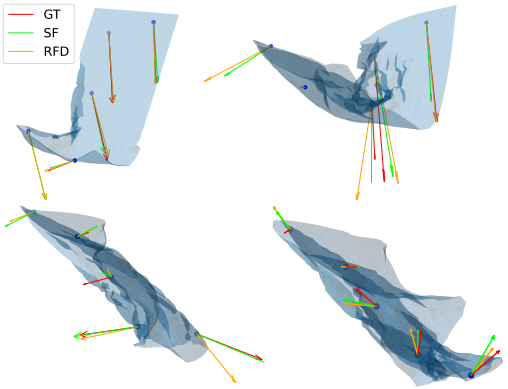

Figure 5. Snapshots of meshes for the velocity prediction task comparing results of our GFI methods with ground truth (**GT**). In several cases, predictions and ground truth are close enough that the velocity vectors appear on top of each other.

Velocity prediction. We further evaluate our algorithms on the deformable `flag_simple` dataset from (Pfaff et al., 2020). The largest mesh sizes from that dataset are of order ~ 1.5k nodes; thus, one can, in principle, apply brute force methods. Therefore this dataset was used only to provide a vision-based validation of the techniques. Fig. 5 shows four sample snapshots of the mesh. The vertex location $\mathbf{x}_i \in \mathbb{R}^3$ and velocity $\mathbf{F}_i \in \mathbb{R}^3$ from each node $\mathbf{n}_i$ in the snapshot are used for interpolation. We randomly mask out 5% of the nodes in each mesh and do a similar interpolation for vertex normals. In the supplementary material, we provide videos representing the dynamics of the deformable meshes and their corresponding fields (ground truth and predicted).

3.2. Wasserstein Distances and Barycenters

Optimal transport (OT) has found many applications in machine learning for its principled approach to compare distributions (Cuturi, 2013). There has been considerable work in extending OT problems to non-Euclidean domains like manifolds (Solomon et al., 2015) and graph-structured data (Mémoli, 2011). Our proposed methods can be easily integrated into several popular algorithms for computing Wasserstein distances. Here, we show the computational efficiency of our algorithms against well-known baselines.

Table 2. Comparison of the total runtime and mean-squared error (MSE) across several meshes for diffusion-based integration. Runtimes are reported in seconds. The lowest time for each mesh is shown in bold. MSE is calculated w.r.t. the output of BF.

Mesh	$\|V\|$	**Total Runtime**		**MSE**
		BF	RFD	
Alien	5212	8.06	**0.39**	0.041
Duck	9862	45.36	**1.10**	0.002
Land	14738	147.64	**2.17**	0.017
Octocat	18944	302.84	**3.36**	0.027

Table 3. Setup as in Table 2, but for the SF algorithm.

Mesh	$\|V\|$	**Total Runtime**		**MSE**
		BF	SF	
Dice	4468	6.8	**4.9**	0.063
Duck	9862	39.2	**19.4**	0.002
Land	14738	90.7	**38.9**	0.015
bubblepot2	18633	113.2	**48.3**	0.081

Wasserstein barycenter. In this section, we consider the OT problem of moving masses on a surface mesh, particularly the computation of Wasserstein barycenters. Since the geodesic distance on a surface is intractable, we use two approximations of this metric: 1) shortest-path distance (used in SF calculations), and 2) distance coming from an ϵ-NN graph approximating the surface (RFD).

Wasserstein barycenter is a weighted average of probability distributions. More formally, given input distributions

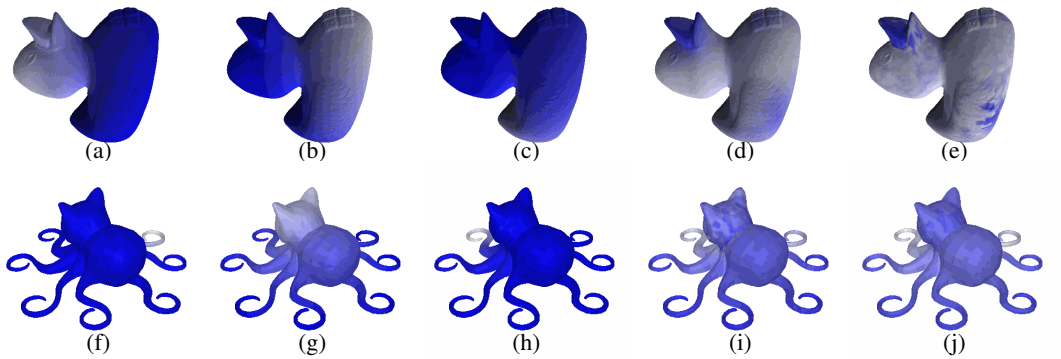

Figure 6. Comparison of the Wasserstein Barycenter output. **a-c, f-h**: three input distributions; **d,i**: Wasserstein Barycenter output with brute-force (BF); **e**: Wasserstein Barycenter output with RFD; **j**: Wasserstein Barycenter output with SF. The top row is for the mesh *duck* and the second row is for the mesh *Octocat-v1*.

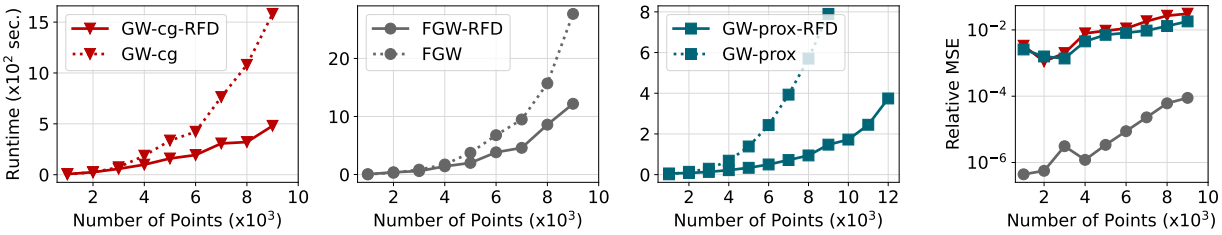

Figure 7. **Left 3 figures**: Plots showing runtimes (in seconds) of *GW* and *FGW* vs our RFDiffusion injected counterparts. **Right figure**: The relative error of our RFDiffusion injected *GW* variants. Except for the *GW*-proximal-RFD, all other variants are OOM after 9k points. For the *FGW* experiments, random binary labels are generated for each node.

$\{\boldsymbol{\mu}^i\}_{i=1}^k$ and a set of weights $\boldsymbol{\alpha} \in \mathbb{R}_+^k$, the Wasserstein barycenter is the solution to the following problem (Agueh & Carlier, 2011):

$$\underset{\boldsymbol{\mu} \in \mathrm{Prob(V)}}{\text{minimize}} \quad \sum_{i=1}^{k} \boldsymbol{\alpha}_i \mathcal{W}_2^2 \left(\boldsymbol{\mu}, \boldsymbol{\mu}^i\right),$$

where $\mathcal{W}_2(\cdot, \cdot)$ denotes the 2-Wasserstein distance.

Algorithm 2 outlined in (Solomon et al., 2015) is used for the experiments in this subsection. We modify their algorithm to directly materialize and plug in our appropriate kernel matrix (which we refer to as the BF algorithm). More details about the baselines are provided in Appendix D.1.2. We give a detailed description of the adaptation of our Fast Multiplication (FM) techniques to the entropic Wasserstein distances (see Appendix D.1.1 and Algorithm 1). Our FM-infused variants significantly speed up the runtime of the baseline algorithm without losing accuracy (see Table 2 and 3.) Three different input probability distributions, encoded as vectors of length $N = |\mathrm{V}|$, are chosen for all our experiments. The barycenter output is also encoded as a vector of length N. Given an estimator $\hat{\boldsymbol{\mu}}$ and the ground truth $\boldsymbol{\mu}$, we measure the quality of the estimator using the mean squared error (MSE) given by $\frac{1}{N}\sum_{j=1}^{N}\left(\hat{\boldsymbol{\mu}}_j - \boldsymbol{\mu}_j\right)^2$.

We visualize the Wasserstein barycenter generated by RFD and SF with their corresponding ground truth (generated by the baseline method) in Fig. 6. Note that the barycenters generated by our integrators are similar to the ground truth. Low-distortion trees do not scale to mesh sizes considered here. We provide additional comparisons of our method with (Solomon et al., 2015) in Appendix (Table 5). However, we note that the results are not directly comparable as the kernels employed by the authors are different from ours.

Gromov Wasserstein and Fused Gromov Wasserstein distances. Gromov Wasserstein (*GW*) discrepancy (resp. Fused Gromov Wasserstein discrepancy (*FGW*)) is an extension of Wasserstein distances to graph-structured data (resp. labeled graph-structured data) with widespread applications in a range of tasks including clustering, classification and generation (Peyré et al., 2016; Mémoli, 2011; Demetci et al., 2020; Mémoli & Sapiro, 2006; Vayer et al., 2018; Titouan et al., 2019). Despite their widespread use, *GW* and *FGW* discrepancies are very expensive to compute.

The *GW* discrepancy can be calculated iteratively by the conditional gradient method (Peyré et al., 2016), which we refer to as *GW*-cg or the proximal point algorithm (Xu et al., 2019), resp. *GW*-prox. A key component in solving this OT problem by either method involves the computation of a tensor product, which is expensive. Our fast GFI methods can be used to estimate this tensor product efficiently (Appendix Algorithm 2), thus speeding up the runtime of the entire algorithm. Moreover, we can also effectively estimate the step size of the *FGW* iterations in a line search algorithm (Appendix Algorithm 3). More details are presented in Appendix D.2.

Since this task is particularly challenging from the computational standpoint, we choose our fastest RFD algorithm. Our methods (*GW*-RFD, *FGW*-RFD and *GW*-prox-RFD) run consistently 2-4x faster than the baseline methods, with only a small drop in accuracy in computing the associated costs (Fig. 7). The plots shown are obtained by averaging over 10 seeds and random 3-D distributions. For all our experiments, $m = 16$ random features, $\epsilon = 0.3$, and the smoothing factor $\lambda = -0.2$ are chosen. For the baseline experiments, we use the implementation from the POT library (Flamary et al., 2021) for the *GW*-cg and *FGW* variants, and official implementation from (Xu et al., 2019) for the *GW*-prox variant. Additional experiments (with ablation studies on the hyperparameters) are in Sec. D.2.4 (resp. E.2).

3.3. Experiments on Point Cloud Classification

In this subsection, we demonstrate the effectiveness of the RFD kernel for various point cloud classification tasks.

Topological Transformers. We present additional experiments with results on Point Cloud Transformers (PCT) (Guo et al., 2021). The entrance point for the RFDiffusion algorithm is the topologically-modulated performized version (Choromanski et al., 2021) of the regular PCT. The topological modulation works by Hadamard-multiplying regular attention matrix with the mask-matrix encoding relative distances between the points in the $3D$ space to directly impose structural priors while training the attention model. Performized PCT provides computational gains for larger point clouds ($N = 2048$ points are used in our experiments). Moreover, its topologically modulated version can be executed in the favorable sub-quadratic time only if the mask-matrix itself supports sub-quadratic matrix-vector multiplication (Choromanski et al., 2022) without the explicit materialization of the attention and the mask matrices. RFDiffusion provides a low-rank decomposition via its novel RF-mechanism and the observation in Sec. 3.4 ((Choromanski et al., 2022)), can be used for time-efficient training in our particular setting. We conduct our experiments on the ShapeNet dataset (Wu et al., 2015). Performized PCT with efficient RFDiffusion-driven masking achieves $\mathbf{91.13}\%$ validation accuracy and linear time complexity (due to the efficient integration algorithm with the RFDiffusion). The brute-force variant runs out of memory in training.

Point Cloud Classification. We have also conducted point cloud classification experiments on ModelNet10 (Wu et al., 2015) and Cubes (Hanocka et al., 2019), using our RFDiffusion kernel method. The classification in this case is conducted using the eigenvectors of the kernel matrix. Note that, as described in (Nakatsukasa, 2019), low-rank decomposition of the kernel matrix (provided directly by the RFDiffusion method via the random feature map mechanism) can be used to compute efficiently eigenvectors and the corresponding eigenvalues. For each dataset, we compute the k smallest eigenvalues of the kernel matrix ($k = 32$ for ModelNet10 and $k = 16$ for Cubes). We pass these k eigenvalues to a random forest classifier for downstream classification. For all the experiments we use: $\epsilon = .1$, $\lambda = -.1$ and we sample 2048 points randomly for each shape in ModelNet10.

The brute-force baseline version for the ModelNet10 and Cubes explicitly constructs the epsilon-neighborhood graph, directly conducting the eigendecomposition of its adjacency matrix and exponentiating eigenvalues. Comparison with this variant is the most accurate apple-to-apple comparison. The baseline variant has time complexity $O(N^3)$ whereas our method for obtaining the eigenvectors is of time complexity $O(N)$. Our results are summarized in Table 4. Our method excels at these point cloud classification tasks, beating the brute-force baseline method by almost $\mathbf{25}$ points on ModelNet10 and by $\mathbf{5}$ points on the Cubes. Our reported numbers are comparable to earlier methods on ModelNet (SPH and LFD achieving 79% (Wu et al., 2015)). Cubes is a fairly challenging dataset and deep learning models like PointNet achieves only 55% accuracy.

Table 4. Point cloud classification using RFD Kernel

Dataset	# Graphs	# Classes	Baseline	RFD
ModelNet10	3991/908	10	43.0	$\mathbf{70.1}$
Cubes	3759/659	23	39.3	$\mathbf{44.6}$

For additional experiments on graph classification, see Appendix F.

4. Conclusion

We have presented in this paper two algorithms, SeparatorFactorization and RFDiffusion, for efficient graph field integration based on the theory of balanced separators and Fourier analysis. As a byproduct of the developed techniques, we have obtained new results in structural graph theory. Our extensive empirical studies support our theoretical findings (e.g., mesh dynamics modeling) involving interpolation on meshes for rigid and deformable objects and the computation of the Wasserstein distance between distributions defined on meshes.

References

Abraham, I., Bartal, Y., and Neiman, O. Nearly tight low stretch spanning trees. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pp. 781–790. IEEE Computer Society, 2008.

Agueh, M. and Carlier, G. Barycenters in the Wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2): 904–924, 2011.

Al-Mohy, A. H. and Higham, N. J. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31(3):970–989, 2010.

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

Avron, H., Kapralov, M., Musco, C., Musco, C., Velingker, A., and Zandieh, A. Random Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.

Bader, P., Blanes, S., and Casas, F. Computing the matrix exponential with an optimized taylor polynomial approximation. *Mathematics*, 7(12):1174, 2019.

Balcilar, M., Renton, G., Heroux, P., Gauzere, B., Adam, S., and Honeine, P. Bridging the gap between spectral and spatial domains in graph neural networks, 2020.

Bartal, Y. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 184–193. IEEE, 1996.

Bartal, Y., Fandina, O. N., and Neiman, O. Covering metric spaces by few trees. *J. Comput. Syst. Sci.*, 130:26–42, 2022.

Barth, T. J. Computational fluid dynamics, structural analysis and mesh partitioning techniques - introduction. In Palma, J. M. L. M., Dongarra, J. J., and Hernández, V. (eds.), *Vector and Parallel Processing - VECPAR '98, Third International Conference, Porto, Portugal, June 21-23, 1998, Selected Papers and Invited Talks*, volume 1573 of *Lecture Notes in Computer Science*, pp. 171–175. Springer, 1998.

Bartolucci, S., Caccioli, F., Caravelli, F., and Vivo, P. Inversion-free Leontief inverse: statistical regularities in input-output analysis from partial information, 2020. URL https://arxiv.org/abs/2009.06350.

Bodlaender, H. L., Drange, P. G., Dregi, M. S., Fomin, F. V., Lokshtanov, D., and Pilipczuk, M. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi: 10.1137/130947374. URL https://doi.org/10.1137/130947374.

Charikar, M., Chekuri, C., Goel, A., Guha, S., and Plotkin, S. A. Approximating a finite metric by a small number of tree metrics. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pp. 379–388. IEEE Computer Society, 1998.

Cheng, H., Crutchfield, W. Y., Gimbutas, Z., Greengard, L., Ethridge, J. F., Huang, J., Rokhlin, V., Yarvin, N., and Zhao, J. A wideband fast multipole method for the Helmholtz equation in three dimensions. *J. Comput. Phys.*, 216(1):300–325, 2006.

Choromanski, K., Lin, H., Chen, H., Zhang, T., Sehanobish, A., Likhosherstov, V., Parker-Holder, J., Sarlós, T., Weller, A., and Weingarten, T. From block-Toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked transformers. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 3962–3983. PMLR, 2022.

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

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 2292–2300, Red Hook, NY, USA, 2013. Curran Associates Inc.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms*. Springer, 2015.

Darve, E. and Have, P. A fast multipole method for Maxwell equations stable at all frequencies. *Philos Trans A Math Phys Eng Sci.*, 15:603–28, 2004. doi: 10.1098/rsta.2003.1337.

Dattoli, G., Ricci, P. E., and Pacciani, P. Comments on the theory of Bessel functions with more than one index. *Appl. Math. Comput.*, 150(3):603–610, 2004.

de Lara, N. and Pineau, E. A simple baseline algorithm for graph classification. *Relational Representation Learning Workshop, NIPS 2018*, 2018.

Demetci, P., Santorella, R., Sandstede, B., Noble, W. S., and Singh, R. Gromov-Wasserstein optimal transport to align single-cell multi-omics data. *bioRxiv*, 2020. doi: 10.1101/2020.04.28.066787.

Diestel, R. and Müller, M. Connected tree-width. *CoRR*, abs/1211.7353, 2012.

Fakcharoenphol, J., Rao, S., and Talwar, K. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

Fellows, M. R., Fomin, F. V., Lokshtanov, D., Losievskaja, E., Rosamond, F. A., and Saurabh, S. Parameterized low-distortion embeddings - graph metrics into lines and trees. *CoRR*, abs/0804.3028, 2008.

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., Tavenard, R., Tong, A., and Vayer, T. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

Gilbert, J. R., Hutchinson, J. P., and Tarjan, R. E. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5 (3):391–407, 1984.

Gimbutas, Z. and Rokhlin, V. A generalized fast multipole method for nonoscillatory kernels. *SIAM J. Sci. Comput.*, 24(3):796–817, 2003.

Greengard, L. and Rokhlin, V. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

Greengard, L., O'Neil, M., Rachh, M., and Vico, F. Fast multipole methods for the evaluation of layer potentials with locally-corrected quadratures. *J. Comput. Phys. X*, 10:100092, 2021.

Gumerov, N. A. and Duraiswami, R. Fast multipole accelerated boundary element methods for room acoustics. *CoRR*, abs/2103.16073, 2021.

Guo, M.-H., Cai, J.-X., Liu, Z.-N., Mu, T.-J., Martin, R. R., and Hu, S.-M. Pct: Point cloud transformer. *Computational Visual Media*, 7(2):187–199, Apr 2021. ISSN 2096-0662. doi: 10.1007/s41095-021-0229-5. URL http://dx.doi.org/10.1007/s41095-021-0229-5.

Han, X., Gao, H., Pfaff, T., Wang, J., and Liu, L. Predicting physics in mesh-reduced space with temporal attention. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. Meshcnn: A network with an edge. *ACM Trans. Graph.*, 38(4), jul 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322959. URL https://doi.org/10.1145/3306346.3322959.

Ishiyama, T., Yoshikawa, K., and Tanikawa, A. High performance gravitational n-body simulations on supercomputer fugaku. In *HPC Asia 2022: International Conference on High Performance Computing in Asia-Pacific Region, Virtual Event, Japan, January 12 - 14, 2022*, pp. 10–17. ACM, 2022.

Janati, H., Cuturi, M., and Gramfort, A. Debiased Sinkhorn barycenters. In *ICML 2020 - 37th International Conference on Machine Learning*, Vienna / Virtuel, Austria, July 2020. URL https://hal.science/hal-03063875.

Li, B. and Chang, G. J. Dot product dimensions of graphs. *Discret. Appl. Math.*, 166:159–163, 2014.

Liska, S. and Colonius, T. A parallel fast multipole method for elliptic difference equations. *J. Comput. Phys.*, 278: 76–91, 2014.

Lozano-Durán, A. and Borrell, G. Algorithm 964: An efficient algorithm to compute the genus of discrete surfaces and applications to turbulent flows. *ACM Trans. Math. Softw.*, 42(4):34:1–34:19, 2016.

Mémoli, F. Gromov—Wasserstein distances and the metric approach to object matching. *Found. Comput. Math.*, 11 (4):417–487, aug 2011. ISSN 1615-3375.

Mémoli, F. and Sapiro, G. Computing with point cloud data. In Krim, H. and Yezzi, A. (eds.), *Statistics and Analysis of Shapes*, pp. 201–229, Boston, MA, 2006. Birkhäuser Boston.

Möller, N., Petit, E., Carayol, Q., Dinh, Q., and Jalby, W. Scalable fast multipole method for electromagnetic simulations. In *Computational Science - ICCS 2019 - 19th International Conference, Faro, Portugal, June 12-14, 2019, Proceedings, Part II*, volume 11537 of *Lecture Notes in Computer Science*, pp. 663–676. Springer, 2019.

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

Mory, B., Ardon, R., Yezzi, A. J., and Thiran, J. Non-euclidean image-adaptive radial basis functions for 3d interactive segmentation. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pp. 787–794. IEEE Computer Society, 2009.

Musco, C., Musco, C., and Sidford, A. Stability of the Lanczos method for matrix function approximation. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans,*

LA, USA, January 7-10, 2018, pp. 1605–1624. SIAM, 2018.

Nakatsukasa, Y. The low-rank eigenvalue problem, 2019.

Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, jan 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13225. URL https://doi.org/10.1613/jair.1.13225.

Orecchia, L., Sachdeva, S., and Vishnoi, N. K. Approximating the exponential, the Lanczos method and an $\tilde{o}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pp. 1141–1160. ACM, 2012.

Pettie, S. and Ramachandran, V. An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, 49 (1):16–34, 2002.

Peyré, G., Cuturi, M., and Solomon, J. Gromov-Wasserstein Averaging of Kernel and Distance Matrices. In *ICML 2016*, Proc. 33rd International Conference on Machine Learning, New-York, United States, June 2016.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pp. 1177–1184. Curran Associates, Inc., 2007.

Rahimi, A. and Recht, B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pp. 1313–1320. Curran Associates, Inc., 2008.

Scetbon, M., Peyré, G., and Cuturi, M. Linear-time Gromov Wasserstein distances using low rank couplings and costs, 2021.

Seenappa, M. G., Potika, K., and Potikas, P. Graph classification with kernels, embeddings and convolutional neural networks. In *2019 First International Conference on Graph Computing (GC)*, pp. 88–93, 2019. doi: 10.1109/GC46384.2019.00021.

Shrivastava, A. and Li, P. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pp. 2321–2329, 2014.

Smola, A. J. and Kondor, R. Kernels and regularization on graphs. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, volume 2777 of *Lecture Notes in Computer Science*, pp. 144–158. Springer, 2003.

Solomon, J., De Goes, F., Peyré, G., Cuturi, M., Butscher, A., Nguyen, A., Du, T., and Guibas, L. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (ToG)*, 34(4):1–11, 2015.

Takahashi, T., Chen, C., and Darve, E. Parallelization of the inverse fast multipole method with an application to boundary element method. *Comput. Phys. Commun.*, 247, 2020.

Thorup, M. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In Larmore, L. L. and Goemans, M. X. (eds.), *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pp. 149–158. ACM, 2003. doi: 10.1145/780542.780566. URL https://doi.org/10.1145/780542.780566.

Titouan, V., Courty, N., Tavenard, R., Laetitia, C., and Flamary, R. Optimal transport for structured data with application on graphs. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6275–6284. PMLR, 09–15 Jun 2019.

Vayer, T., Chapel, L., Flamary, R., Tavenard, R., and Courty, N. Fused Gromov-Wasserstein distance for structured objects: theoretical foundations and mathematical properties, 2018.

Villani, C. *Topics in optimal transportation*. Graduate studies in mathematics ; v. 58. American Mathematical Society, Providence, R.I, 2003. ISBN 082183312X.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition*, 2015.

Xu, H., Luo, D., Zha, H., and Duke, L. C. Gromov-Wasserstein learning for graph matching and node embedding. In Chaudhuri, K. and Salakhutdinov, R. (eds.),

Proceedings of the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, pp. 6932–6941. PMLR, 09–15 Jun 2019.

Yokota, R., Ibeid, H., and Keyes, D. E. Fast multipole method as a matrix-free hierarchical low-rank approximation. *CoRR*, abs/1602.02244, 2016.

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

Appendix: Efficient Graph Field Integrators Meet Point Clouds

Broader Impacts

Matrix-vector multiplication is a core component of all machine learning (ML) models. Thus there is a lot of interest in the ML community to discover ways or use cases where the above operation can be done in an efficient manner. This problem of fast matrix-vector multiplication also has tremendous applications in physical sciences, chemistry, and networking protocols. A vast body of literature has proposed scenarios where this problem is applicable. Our work makes an important contribution towards this direction by discovering new examples where such methods exist. We expect our work to benefit the ML community and the broader scientific community. Our work is mostly theoretical in nature, therefore we do not foresee any negative applications of our algorithms.

A. Theoretical Analysis

A.1. Warmup Results on (G, f)-tractability

We start with the following simple remark:

Remark A.1. Let $\mathcal{G}$ be a family of graphs and let $\mathcal{F} = \{f_1, \ldots, f_{|\mathcal{F}|}\}$ be a family of functions $\mathbb{R} \rightarrow \mathbb{C}$. If $(\mathcal{G}, f_i)$ is T-tractable for $i = 1, \ldots, |\mathcal{F}|$ then for any $f : \mathbb{R} \rightarrow \mathbb{C}$ of the form: $f(z) = \sum_{i=1}^{|\mathcal{F}|} a_i f_i(z)$, where $a_1, \ldots, a_{\mathcal{F}} \in \mathbb{C}$, $(\mathcal{G}, f)$ is $(T \cdot \mathcal{F})$-tractable.

Furthermore, the following trivially holds:

Remark A.2. Let $\mathcal{G}$ be a family of graphs and let $f : \mathbb{R} \rightarrow \mathbb{C}$ be a function. If $(\mathcal{G}, f)$ is T-tractable then $(\mathcal{G}, \mathrm{Re}(f))$ and $(\mathcal{G}, \mathrm{Im}(f))$ are T-tractable, where Re and Im stand for the real and imaginary part of f respectively.

The $|\mathrm{V}|$-tractability of $(\mathcal{T}, f)$, where $\mathcal{T}$ is the family of trees and $f(z) = \exp(az + b)$, proven in (Choromanski et al., 2022), combined with the above remarks implies several results for specific important classes of functions f. In particular, the following holds:

Corollary A.3. *If $\mathcal{T}$ is the family of trees and f is given by a finite Fourier series of length L, then $(\mathcal{T}, f)$ is $(\mathrm{V} \cdot L)$-tractable. Thus in particular: $(\mathcal{T}, f)$ is $|\mathrm{V}|$-tractable for $f(z) = A\sin(\omega z + \phi)$ for $A, \omega, \phi \in \mathbb{R}$. This remain true if $f(z) = A\exp(-bz)\sin(\omega z + \phi)$, where $b \in \mathbb{R}$.*

A.2. Proof of Theorem 2.4

Proof. Let $\mathrm{G} \in \mathcal{G}$ and denote: $N = |\mathrm{V}(\mathrm{G})|$. Without loss of generality, we can assume that G is connected. We start with the following auxiliary lemma, where we introduce the key notion of graph *separator*:

Lemma A.4. *The set of vertices $\mathrm{V}(\mathrm{G})$ of G can be partitioned in time $O(N)$ into three pairwise disjoint sets: $\mathcal{A}, \mathcal{B}, \mathcal{S}$ such that: $\delta N \leq |\mathcal{A}|, |\mathcal{B}| \leq (1 - \delta)N$ for some universal $0 < \delta < 1$ and $|\mathcal{S}| \leq t + 1$, where $t = \mathrm{ctw}(\mathrm{G})$ stands for the connected treewidth of G. Furthermore, no edges exist between $\mathcal{A}$ and $\mathcal{B}$ and: the induced sub-graphs $\mathrm{G}_{\mathcal{A}}$ and $\mathrm{G}_{\mathcal{B}}$ of G with sets of vertices $\mathcal{A}$ and $\mathcal{B}$ respectively both have connected treewidth at most $\mathrm{ctw}(\mathrm{G})$. Finally, the graph $\mathrm{G}_{\mathcal{S}}$ induced by $\mathcal{S}$ is connected. We call set $\mathcal{S}$ a separator in G. Furthermore, the tree decomposition with connected bags and of treewidth t can be found in time $O(N)$, with $\mathcal{S}$ being one of the bags.*

Proof. This follows directly from the algorithmic proof of the following theorem:

Theorem A.5 (Bodlaender et al., 2016)**.** *For a graph on N vertices with treewidth k, there is an algorithm that will return a tree decomposition with width $5k + 4$ in time $2^{O(k)}N$.*

The bounded treewidth decomposition from the above theorem can be easily refined to the bounded connected treewidth deecomposition. Note that the $O(N)$ time-complexity algorithm for the bounded connected treewidth decomposition immediately implies that its representation is of size $O(N)$ (i.e. the corresponding tree has $O(N)$ edges/vertices).

Since our graph admits a tree decomposition with connected bags, the above tree decomposition can also be constructed to have this property. Now we can apply the algorithmic version of the proof of Lemma 7.19 from (Cygan et al., 2015),

concluding that one of the bags of this tree decomposition is a balanced separator and can be found by searching the tree in time $O(N)$.

The base version of the lemma is provided below:

Lemma A.6. *Assume that* G *is a graph of treewidth at most* k*. Then there exists a separator* X *in* G *of size at most* $k+1$ *and such that each connected component of the graph obtained from* G *by deleting vertices from* X *and the incident edges has at most* $\frac{1}{2}|V(\mathrm{G})|$ *vertices.*

Let us explain know how the algorithmic version of the proof works. From the proof of the Lemma A.6 it is clear that as long as computing the sizes of the sets for all nodes of the tree (from the treewidth decomposition) can be done in $O(N)$ time, the balanced separator can be found in $O(N)$ time (V_t in the Lemma refers to the union of all the bags in the nodes of the tree rooted in t). However this can be done via a standard recursion algorithm. Consider a node t which is not a leaf and its children: $c_1, c_2, \ldots$ Note that in order to compute the size of the union of the sets: $V_{c_1}, V_{c_2}, \ldots$, all that we need is: (a) the individual sizes of the sets: $V_{c_1}, V_{c_2}, \ldots$ (that can be stored in the individual nodes as we progress with the recursion) (b) the number of children from the set $\{c_1, c_2, \ldots\}$ whose corresponding bags contain a given vertex x from the bag B_t associated with t (for every x in B_t). This is true since sets $V_{c_1}, V_{c_2}, \ldots$ are not necessarily disjoint, but by the definition of the treewidth, their intersections are subsets of B_t. Furthermore, if V_{c_i} intersects with B_t, then (again, directly from the treewidth definition) this intersection is also a subset of the bag B_{c_i} corresponding to c_i. All the computations from (b) can be clearly done in time $s \times O(k^2)$, where s is the number of children of t and k is an upper bound on the bag size. Since we consider bounded connected treewidth graphs, time complexity reduces to $O(s)$. By unrolling this recursion, we obtain the algorithm of time complexity proportional to the number of edges of the tree from the treewidth decomposition which is $O(N)$ (see our discussion above). That completes the analysis.

□

We are ready to prove the Theorem. We will identify the set of vertices $\mathrm{V}(\mathrm{G})$ with the set $\{1, \ldots, N\}$. Denote:

$$v_i = \sum_{j=1}^{N} f(\mathrm{dist}(i,j)) x_j \tag{15}$$

For a subset $\mathrm{S} \subseteq \{1, \ldots, N\}$, we will also use the following notation:

$$v_i^{\mathrm{S}} = \sum_{j \in \mathrm{S}} f(\mathrm{dist}(i,j)) x_j \tag{16}$$

Our goal is to compute v_i for $i = 1, \ldots, N$. Equipped with Lemma A.4, we find the decomposition of $\mathrm{V}(\mathrm{G})$ into $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{S}$ in time $O(N)$. We find the entire tree decomposition $\mathcal{T}$ from Lemma A.4. Our strategy is first to compute v_i for all $i \in \mathcal{S}$ and then to compute v_i for all $i \in \mathcal{A} \cup \mathcal{B}$.

1. Computation of v_i for $i \in \mathcal{S}$.
For every $i \in \mathcal{S}$, we can simply run Dijkstra's algorithm (or one of its improved variants mentioned in the main body) to find shortest path trees and, consequently, compute quantities v_i. This can clearly be done in time $O(|\mathcal{S}|(N+M)\log(M))$, where M is the number of edges of G. Since G is sparse, the total time complexity is $O(N\log(N))$ (or even $O(N\log\log(N))$ if the fastest algorithms for finding shortest paths in graphs with positive weights are applied, see: (Thorup, 2003)).

2. Computation of v_i for $i \in \mathcal{A} \cup \mathcal{B}$.
We will show how to compute v_i for all $i \in \mathcal{A}$. The calculations of v_i for all $i \in \mathcal{B}$ will be completely analogous. Note first that:

$$v_i = v_i^{\mathcal{A}} + v_i^{\mathcal{S} \cup \mathcal{B}} \tag{17}$$

2.1 Computation of $v_i^{\mathcal{A}}$ for all $i \in \mathcal{A}$. We first show how to compute $v_i^{\mathcal{A}}$ for all $i \in \mathcal{A}$. Take a vertex $j \in \mathcal{A}$. Denote by $P_{i,j}$ the shortest path from i to j in G. If $P_{i,j}$ contains vertices from $\mathcal{B}$ then $P_{i,j}$ needs to use some vertices from $\mathcal{S}$ (since there are no edges between $\mathcal{A}$ and $\mathcal{B}$). If that is the case, denote by x the first vertex from $\mathcal{S}$ on the path $P_{i,j}$ as we go from i to j and by y the last vertex from $\mathcal{S}$ on the path $P_{i,j}$ as we go from i to j. Note that $x \neq y$. Denote by $P_{x,y}$ the

sub-path of the path $P_{i,j}$ that starts at x and ends at y. Note that all the vertices of $\mathcal{B}$ that belong to $P_{i,j}$ also belong to $P_{x,y}$. Furthermore, since $P_{i,j}$ is the shortest path from i to j, $P_{x,y}$ is the shortest path from x to y.

Note that $P_{x,y}$ is of length at most $|\mathcal{S}|-1$, where $|\mathcal{S}|$ is the size of $\mathcal{S}$. This is the case since there exists a path from x to y using only vertices from $\mathcal{S}$ (since $\mathrm{G}_{\mathcal{S}}$ is connected). The number of edges $m_{x,y}$ of the path $P_{x,y}$ using at least one vertex from $\mathcal{B}$ is at most t.

Denote: $\mathcal{P}=\{P_{x,y}: x,y\in\mathcal{S}, x\neq y\}$, where $P_{x,y}$ is the shortest path from x to y (if there are many such paths, choose an arbitrary one). Note that the size of $\mathcal{P}$ satisfies: $|\mathcal{P}|\leq\binom{|\mathcal{S}|}{2}=\binom{t+1}{2}$. Denote by $\mathcal{B}'$ the subset of $\mathcal{B}$ consisting of the vertices of $\mathcal{B}$ that belong to these paths from $\mathcal{P}$ that have length at most t. Note that the size of $\mathcal{B}'$ satisfies: $|\mathcal{B}'|\leq t\binom{t+1}{2}$. Furthermore, set $\mathcal{B}'$ can be found in time $O(|\mathcal{S}|(N+M)\log(M))=O(N\log(N))$, simply by running Dijkstra algorithm for every vertex: $x\in\mathcal{S}$ (again, as before, this time can be improved to $N\log\log(N)$).

Note that since G has bounded connected treewidth, a subset $\mathcal{C}\subseteq\mathcal{B}$ of constant size can be found in $O(N)$ time (using $\mathcal{T}$) such that: $\mathrm{G}[\mathcal{A}\cup\mathcal{S}\cup\mathcal{B}'\cup\mathcal{C}]$ has bounded connected treewidth t. Denote: $\mathcal{A}_{\mathrm{ext}}=\mathcal{A}\cup\mathcal{S}\cup\mathcal{B}'\cup\mathcal{C}$. By the definition of $\mathcal{B}'$, for every $i,j\in\mathcal{A}$ at least one of the shortest paths between i and j lies entirely in the sub-graph of G induced by $\mathcal{A}_{\mathrm{ext}}$. Furthermore, the sub-graph $\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}}$ induced by $\mathcal{A}_{\mathrm{ext}}$ has connected treewidth at most $\mathrm{ctw}(\mathrm{G})$. We then recursively compute for each $i\in\mathcal{A}$ the following expression:

$$\tilde{v}_i=\sum_{j\in\mathrm{V}(\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}})}f(\mathrm{dist}_{\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}}}(i,j))x_j, \tag{18}$$

where $\mathrm{dist}_{\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}}}$ is the shortest path distance in graph $\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}}$.

Note that we have: $v_i=\tilde{v}_i-\delta_i$, where:

$$\delta_i=\sum_{j\in\mathcal{S}\cup\mathcal{B}'\cup\mathcal{C}}f(\mathrm{dist}_{\mathrm{G}_{\mathcal{A}_{\mathrm{ext}}}}(j,i))x_j \tag{19}$$

All δ_i can be computed in time $O(|\mathcal{S}\cup\mathcal{B}'\cup\mathcal{C}|(N+M)\log(M))=O(N\log(N))$, simply by running Dijkstra's algorithms for every vertex $v\in\mathcal{S}\cup\mathcal{B}'\cup\mathcal{C}$ (as before, this can be improved to $O(N\log\log(N))$ time). That completes the computation of $v_i^{\mathcal{A}}$ for every $i\in\mathcal{A}$.

2.2 Computation of $v_i^{\mathcal{S}\cup\mathcal{B}}$ for all $i\in\mathcal{A}$. It remains to show how to compute $v_i^{\mathcal{S}\cup\mathcal{B}}$ for every $i\in\mathcal{A}$. To do that, we introduce for every vertex $v\in\mathrm{V}(\mathrm{G})$ a vector $\chi_v\in\mathbb{R}^{|\mathcal{S}|}$ defined as follows:

$$\chi_v[k]=\mathrm{dist}(v,k) \tag{20}$$

for $k=1,\ldots,|\mathcal{S}|$. In the above, we identify the set $\mathcal{S}$ with $\{1,\ldots,|\mathcal{S}|\}$. Note that the following is true for any $i\in\mathcal{A}$, $j\in\mathcal{S}\cup\mathcal{B}$:

$$\mathrm{dist}(i,j)=\min_{k\in\mathcal{S}}\left(\chi_i[k]+\chi_j[k]\right), \tag{21}$$

since every path from i to j needs to use a vertex from $\mathcal{S}$. The following is also true:

$$\chi_v=\tau_v+\rho_v, \tag{22}$$

where: $\tau_v[i]=\min_{k\in\mathcal{S}}\mathrm{dist}(v,k),\forall i$ and furthermore ρ_v is a vector satisfying for $k=1,\ldots,|\mathcal{S}|$:

$$0\leq\rho_v[k]\leq|\mathcal{S}|-1=t \tag{23}$$

The latter is true since the lengths of any two shortest paths from v to two vertices of $\mathcal{S}$ differ by at most $|\mathcal{S}|-1$ (because $\mathrm{G}_{\mathcal{S}}$ is connected and thus there exists a path between any two vertices of $\mathrm{G}_{\mathcal{S}}$ of length at most $|\mathcal{S}|-1$). We call ρ_v the *signature* of v with respect to $\mathcal{S}$. The following holds:

$$\mathrm{dist}(i,j)=\tau_i[1]+\tau_j[1]+\min_{k\in\mathcal{S}}(\rho_i[k]+\rho_j[k]) \tag{24}$$

Note that all the vectors ρ_i and τ_i for $i\in\mathrm{V}(\mathrm{G})$ can be computed in time $O(|\mathcal{S}|(N+M)\log(M))=O(N\log(N))$, by running Dijkstra's algorithm for every vertex $v\in\mathcal{S}$ (and again, we can improve this time complexity to $O(N\log\log(N))$).

We then partition the set $\mathcal{A}$ into subsets $\mathcal{A}_\rho$ (some of them potentially empty) indexed by the vectors $\rho \in \{0, 1, \ldots, |\mathcal{S}|-1\}^{|\mathcal{S}|}$:

$$\mathcal{A}_\rho = \{i \in \mathcal{A} : \rho_i = \rho\} \tag{25}$$

We define the partitioning of $\mathcal{C} = \mathcal{S} \cup \mathcal{B}$ into subsets $\mathcal{C}_\rho$ in the analogous way.

We will first compute $v_i^{\mathcal{C}_{\rho^2}}$ for every $i \in \mathcal{A}_{\rho^1}$ for given $\rho^1, \rho^2 \in \{0, 1, \ldots, |\mathcal{S}| - 1\}^{|\mathcal{S}|}$. For fixed $\rho^1, \rho^2 \in \{0, 1, \ldots, |\mathcal{S}| - 1\}^{|\mathcal{S}|}$, we first show how to compute $v_i^{\mathcal{C}_{\rho^2}}$ for all $i \in \mathcal{A}_{\rho^1}$. We partition $\mathcal{A}_{\rho^1}$ into subsets (some of them potentially empty):

$$\mathcal{A}^l_{\rho^1} = \{i \in \mathcal{A}_{\rho^1} : \tau_i[1] = l\} \tag{26}$$

for $l = 0, 1, \ldots, N - 1$.

We define the partitioning of $\mathcal{C}_{\rho^2}$ into subsets $\mathcal{C}^l_{\rho^2}$ in the analogous way.

Note that for $i \in \mathcal{A}^{l_1}_{\rho^1}$ and $j \in \mathcal{C}^{l_2}_{\rho^2}$ the following is true:

$$\text{dist}(i, j) = l_1 + l_2 + g(\rho^1, \rho^2), \tag{27}$$

where $g(\rho^1, \rho^2) \stackrel{\text{def}}{=} \min_{k \in \mathcal{S}}(\rho^1[k] + \rho^2[k])$. We observe that the quantity $v_i^{\mathcal{C}_{\rho^2}}$ is the same for every $i \in \mathcal{A}^{l_1}_{\rho^1}$. Thus it suffices to compute $\{v_{i_0}^{\mathcal{C}_{\rho^2}}, \ldots, v_{i_{N-1}}^{\mathcal{C}_{\rho^2}}\}$ for arbitrary representatives $i_u \in \mathcal{A}^u_{\rho^1}$ (without loss of generality, we will assume that all sets $\mathcal{A}^u_{\rho^1}$ for $u = 0, \ldots, N - 1$ are nonempty). If we define vector $\mathbf{w} \in \mathbb{R}^N$ as: $\mathbf{w}[u] = v_{i_u}^{\mathcal{C}_{\rho^2}}$ then we have: $\mathbf{w} = \mathbf{W}\mathbf{z}$, where vector $\mathbf{z} \in \mathbb{R}^N$ is given as follows:

$$\mathbf{z}[u] = \sum_{v \in \mathcal{C}^u_{\rho^2}} x_v \tag{28}$$

and furthermore matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ is given as:

$$\mathbf{W}[l_1, l_2] = f(l_1 + l_2 + g(\rho^1, \rho^2)) \tag{29}$$

Vector $\mathbf{z}$ can be easily computed in $O(N)$ time. The key observation is that multiplication $\mathbf{W}\mathbf{z}$ can be conducted in $O(N \log(N))$ time (the matrix $\mathbf{W}$ does not need to be explicitly materialized) with the use of Fast Fourier Transform since $\mathbf{W}$ is a *Hankel matrix* (constant along each anti-diagonal). Thus we conclude that we can compute $v_i^{\mathcal{C}_{\rho^2}}$ for all $i \in \mathcal{A}_{\rho^1}$ in $O(N \log(N))$ time. If a kernel is defined as $\mathrm{K}(i, j) = \exp(-\lambda \text{dist}(\mathrm{i}, \mathrm{j}))$, this becomes a very special Hankel matrix, where each row is obtained from the previous one by multiplication with a fixed constant. It is easy to see that the multiplications with such matrices $\mathbf{W}$ can be conducted in time $O(N)$ (we thus save a $\log(N)$-factor).

By applying this method to all pairs (ρ^1, ρ^2), we conclude that we can compute $v_i^{\mathcal{S} \cup \mathcal{B}}$ for all $i \in \mathcal{A}$ in time $O(|\mathcal{S}|^{|\mathcal{S}|-1} \cdot |\mathcal{S}|^{|\mathcal{S}|-1} N \log(N)) = O(N \log(N))$. This can be improved to $O(N)$ time if $\mathrm{K}(i, j) = \exp(-\lambda \text{dist}(\mathrm{i}, \mathrm{j}))$ is being applied.

Combining step 1 with steps 2.1 and 2.2, we obtain a method for computing v_i for every $i \in \mathcal{A}$. The computations of v_i for every $i \in \mathcal{B}$ are conducted in a completely analogous way (where we borrow the notation from the above analysis but adapt to this case, e.g., we replace $\mathcal{B}'$ with $\mathcal{A}'$).

3. Putting this all together – time complexity analysis.
To summarize, v_i for all $i \in \mathrm{V}(\mathrm{G})$ can be computed in time:

$$T(N) = T(N_1) + T(N_2) + O(N \log(N)), \tag{30}$$

where: $\rho N \leq N_1, N_2 \leq (1 - \rho)N + C$ for constants $0 < \rho < 1, C > 0$. It is easy to see that the solution to this recursive equation satisfies the following:

$$T(N) = O(N \log^2(N)) \tag{31}$$

If the kernel being used is of the form: $\mathrm{K}(i, j) = \exp(-\lambda \text{dist}(\mathrm{i}, \mathrm{j}))$, then the recursion for the total runtime is of the form:

$$T(N) = T(N_1) + T(N_2) + O(N \log \log(N)), \tag{32}$$

which implies that: $T = O(N \log^{1.38}(N))$. That completes the entire proof. □

Remark A.7. Note that the proof of the above result but for the family $\mathcal{G}$ of trees from (Choromanski et al., 2022) is a special instantiation of the proof presented above.

A.3. Proof of Lemma 2.6

Proof. Denote: $\tilde{f}(\mathbf{z}) = \int_{\mathcal{B}(R)} \exp(2\pi \mathbf{i} \omega^\top \mathbf{z}) \tau(\omega) d\omega$ for $\mathbf{z} = \mathbf{n}_v - \mathbf{n}_w$. Note that in the efficient implementation of the RFD-estimator (that neglects the imaginary part of the dot-product), we have:

$$\widehat{\mathbf{W}}(v,w) = \frac{1}{m} \sum_{i=1}^{m} X_i, \tag{33}$$

where: $X_i = \cos(2\pi \omega_i^\top \mathbf{z}) \frac{\tau(\omega_i)}{p(\omega_i)}$, τ is the Fourier Transform (FT) of the function $\mathbb{1}[\|\mathbf{z}\|_1 \leq \epsilon]$ and p is the pdf of the R-truncated Gaussian distribution (e.g. Gaussian distribution truncated to the L_1-ball $\mathcal{B}(R)$ of radius R and centered at 0). We clearly have: $\mathbb{E}[\widehat{\mathbf{W}}(v,w)] = \tilde{f}(\mathbf{z})$. Furthermore, the following holds:

$$\mathrm{Var}(\widehat{\mathbf{W}}(v,w)) = \frac{1}{m^2} \cdot m \cdot \mathrm{Var}(X_1) = \frac{1}{m} \mathrm{Var}(X_1) \tag{34}$$

Note that: $p(\omega) = \frac{1}{(2\pi)^{\frac{3}{2}}} \exp(-\frac{\|\omega\|^2}{2}) \cdot C^{-1}$. We have: $\mathrm{Var}(X_1) = \mathbb{E}[X_1^2] - (\tilde{f}(\mathbf{z}))^2$ and:

$$\begin{aligned} \mathbb{E}[X_1^2] = \int_{\mathcal{B}(R)} \cos^2(2\pi \omega^\top \mathbf{z}) \tau^2(\omega) p^{-1}(\omega) d\omega \leq (2\pi)^{\frac{3}{2}} C \int_{\mathcal{B}(R)} \frac{\sin^2(2\epsilon\omega_1)}{\omega_1^2} \cdot \ldots \cdot \frac{\sin^2(2\epsilon\omega_d)}{\omega_d^2} \cdot \\ \exp(\frac{\omega_1^2}{2}) \cdot \ldots \cdot \exp(\frac{\omega_d^2}{2}) d\omega = (2\pi)^{\frac{3}{2}} C \Gamma_\epsilon^d \end{aligned} \tag{35}$$

We have leveraged the formula for the FT of the function $\mathbb{1}[\|\mathbf{z}\|_1 \leq \epsilon]$ from the main body of the paper. Thus we have:

$$\mathrm{Var}(\widehat{\mathbf{W}}(v,w)) \leq \frac{1}{m} \left((2\pi)^{\frac{3}{2}} C (\Gamma_\epsilon(R))^d - (\tilde{f}(\mathbf{z}))^2 \right) \tag{36}$$

We also have:

$$\mathrm{MSE}(\widehat{\mathbf{W}}(v,w)) \leq \mathrm{Var}(\widehat{\mathbf{W}}(v,w)) + (\tilde{f}(\mathbf{z}))^2 - (f(\mathbf{z}))^2 = \mathrm{Var}(\widehat{\mathbf{W}}(v,w)) + (\tilde{f}(\mathbf{z}) - f(\mathbf{z}))(\tilde{f}(\mathbf{z}) + f(\mathbf{z})) \tag{37}$$

Plugging in the formula for $\tilde{f}(\mathbf{z})$ and the variance, we complete the proof. □

B. Graph Metric Approximation with Trees

Define $\mathrm{dist}_{\mathrm{G}}(\cdot,\cdot)$ to be a shortest path distance function on a graph G. Recall that when $\mathcal{T}$ is the family of trees and $f(z) = \exp(az + b)$, then we can compute the GFI in time $O(|\mathrm{V}|)$ using dynamic programming (via single bottom-up and single top-down traversal of the tree). In other words, $(\mathcal{T}, f)$ is $|\mathrm{V}|$-tractable. Therefore, it is advantageous to consider representing a graph by trees that preserve/approximate its metric.

Spanning tree. A naive tree approximating the weighted graph metric is the graph's minimum spanning tree. The optimal running time for finding a minimum spanning tree is $O(|\mathrm{E}| \cdot \alpha(|\mathrm{V}|, |\mathrm{E}|))$ (Pettie & Ramachandran, 2002), where α is incredibly slowly growing function.

Note that while it is cheap to build a spanning tree, the distortion of the shortest path distance of the original graph can be considerable. For example, let G be an unweighted n-cycle and T be its minimum spanning tree. Then the distortion between leaf nodes in the spanning tree is $\mathrm{dist}_{\mathrm{T}}(u,v)/\,\mathrm{dist}_{\mathrm{G}}(u,v) = n - 1$. However, if instead of a single tree, we can embed our graph into a distribution over trees, then we can hope to get better *expected* distortion. In this specific example, if we take a uniform distribution over n different spanning trees (each obtained by deleting an edge), then the expected distance distortion becomes

$$\mathbb{E}_{\mathrm{T}}\left[\mathrm{dist}_{\mathrm{T}}(u,v)/\,\mathrm{dist}_{\mathrm{G}}(u,v)\right] = 2(1 - 1/n).$$

This brings us to another type of method based on embedding the arbitrary weighted graph metric into the distribution of trees.

Low-distortion trees. Building on the low diameter randomized decomposition, Bartal (1996) introduced an algorithm for sampling random hierarchically well-separated trees with expected distortion factor $O(\log^2 |\mathrm{V}|)$. Assume that the diameter of an input graph is $O(\mathrm{poly}(|\mathrm{V}|))$. Then for all $u, v \in \mathrm{V}$, the expectation over random tree T of the distance distortion is

$$\mathbb{E}_{\mathrm{T}}\left[\mathrm{dist}_{\mathrm{T}}(u,v)/\,\mathrm{dist}_{\mathrm{G}}(u,v)\right] \in \left[1,\ O(\log^2 |\mathrm{V}|)\right].$$

Time complexity to sample a single Bartal tree is $O(\log|\mathrm{V}|(|\mathrm{E}| + |\mathrm{V}|\log|\mathrm{V}|))$. Further, Fakcharoenphol et al. (2004) improved an arbitrary metric space embedding into random trees by providing a constructive algorithm with optimal distortion factor of $\Theta(\log|\mathrm{V}|)$, i.e.,

$$\mathbb{E}_{\mathrm{T}}\left[\mathrm{dist}_{\mathrm{T}}(u,v)/\,\mathrm{dist}_{\mathrm{G}}(u,v)\right] \in \left[1,\ O(\log|\mathrm{V}|)\right].$$

Note that this improvement comes at higher time complexity for sampling a tree. Unlike FRT trees (Fakcharoenphol et al., 2004), during the low-diameter decomposition in the Bartal algorithm, the cluster center is always included in the cluster itself. As a result, in the Bartal algorithm, we can consolidate the sub-trees recursively without introducing new vertices. In contrast, the FRT algorithm defines a laminar family, which corresponds to a rooted tree with graph nodes at the leaves.

In our applications for fast graph field integration during preprocessing, we sample $\mathrm{T}_1, \ldots, \mathrm{T}_k$ trees independently from one of the above distributions. Note that sampling can be done in parallel. For the inference, we compute

$$i(v) = \frac{1}{k}\sum_{i=1}^{k}\sum_{w\in\mathrm{V}} f(\mathrm{dist}_{\mathrm{T}_i}(w,v))\mathcal{F}(w),$$

which takes $O(k|\mathrm{V}|)$. The integration on each tree can be carried out in parallel, reducing the inference time by a factor of k.

In Fig. 4, we set the number of trees in T-FRT (FRT trees) to 3 and implemented two variants of T-Bart (Bartal trees) with 3 and 20 trees, respectively.

C. Interpolation on Meshes

In this section, we present implementation details for Sec. 3.1. All experiments are run on a single computer with an i9-12900k CPU and 96GB memory.

C.1. Vertex Normal Prediction.

In the vertex normal prediction task in Sec. 3.1, we choose 120 meshes for 3D-printed objects with a wide range of size from the Thingi10k (Zhou & Jacobson, 2016) dataset with the following File IDs:

```
[60246, 85580, 40179, 964933, 1624039, 91657, 79183, 82407, 91658, 40172, 65414, 90431,
74449, 73464, 230349, 40171, 61193, 77938, 375276, 39463, 110793, 368622, 37326, 42435,
1514901, 65282, 116878, 550964, 409624, 101902, 73410, 87602, 255172, 98480, 57140,
285606, 96123, 203289, 87601, 409629, 37384, 57084, 136024, 202267, 101619, 72896, 103538,
90064, 53159, 127243, 293452, 78671, 75667, 285610, 80597, 90736, 75651, 1220293, 126660,
75654, 75657, 111240, 75665, 75652, 68706, 123472, 88855, 470464, 444375, 208741, 80908,
73877, 495918, 1215157, 85758, 80516, 101582, 75496, 441708, 796150, 257881, 68381,
294160, 265473, 762595, 461110, 461111, 38554, 762594, 79353, 81589, 95444, 762586,
762610, 762607, 1335002, 274379, 437375, 59333, 551074, 550810, 93130, 372053, 372059,
133078, 178340, 133079, 133568, 331105, 80650, 47984, 551021, 308214, 372057, 59197,
1717685, 439142, 372058, 376252, 372114]
```

For each method listed in Fig. 5, we do a grid-search over its hyper-parameter(s) for each mesh and report the pre-processing time and interpolation time associated with the hyper-parameter(s) that give(s) us the best cosine similarity.

D. Wasserstein Distances and Barycenters

The Wasserstein metric has received a lot of attention in the machine learning community, especially for its principled way of comparing distributions on a metric space $\mathcal{X}$ (Villani, 2003). It is a distance function between probability measures defined on $\mathcal{X}$, while strongly reflecting the metric of the underlying space. In spite of their broad use, Wasserstein distances are computationally expensive.

D.1. Wasserstein Barycenters on Meshes

To alleviate this computational bottleneck, convolutional Wasserstein distance is proposed as an entropic regularized Wasserstein distance over geometric domains by leveraging the heat kernel as the proxy for the geodesic distance over the manifold.

The heat kernel matrix $\mathbf{H}$ can be seen as a generalization of a Gaussian kernel on a manifold by Varadhan's formula (Solomon et al., 2015). Moreover, the geodesic distance function $\mathrm{dist}(i,j)$ on a surface mesh can be approximated by the Euclidean distance on an ϵ-nearest-neighbor graph (when ϵ is suitably chosen). Note that, unlike the Gaussian kernel, the heat kernel can be efficiently computed.

In our work, we approximate the geodesic distance on a surface mesh by (1) shortest-path distance (used in SF calculations), and 2) distance coming from an ϵ-NN graph approximating the surface (RFD).

D.1.1. Efficient computation of Wasserstein barycenter

One of the key steps for the computation of Wasserstein distance is the action $\mathbf{H}$ on a given vector $\mathbf{x}$. Solomon et al. (2015) use a pre-factorized decomposition of $\mathbf{H}$ to do the above matrix-vector multiplication efficiently without actually materializing $\mathbf{H}$.

Similar to their method, we never materialize our kernel matrices $\mathbf{K}$ explicitly, i.e. we only need to know how to apply $\mathbf{K}$ to vectors. Here FM can either be SeparationFactorization (SF) or the RFDiffusion algorithm (RFD) and for clarity, we use the subscript for the matrix $\mathbf{K}$ to specify that we are approximating the (right) action of the matrix $\mathbf{K}$.

We define $\otimes$ as the Hadamard product (also known as the element-wise product) and $\oslash$ as the element-wise division.

Algorithm 1 Fast Computation of Wasserstein Barycenter

Inputs: probability distributions $\left\{\boldsymbol{\mu}^i\right\}_{i=1}^k$, area weights $\vec{a} \in \mathbb{R}_+^N$, maxIter $\in \mathbb{N}$, $\boldsymbol{\alpha} \in \mathbb{R}_+^k$
Output: Wasserstein barycenter $\boldsymbol{\mu} \in \mathrm{Prob(V)}$
Initialize: $\mathbf{v}^1, \ldots, \mathbf{v}^k \leftarrow \vec{1}, \mathbf{w}^1, \ldots, \mathbf{w}^k \leftarrow \vec{1}, \boldsymbol{\mu} \leftarrow \vec{1}$.
for $j \leq$ maxIter
  for $i = 1, \ldots, k$
    1. $\mathbf{w}^i \leftarrow \boldsymbol{\mu}^i \oslash \left(\mathrm{FM}_{\mathbf{K}}(\vec{a} \otimes \mathbf{v}^i)\right)$
    2. $\mathbf{d}^i \leftarrow \mathbf{v}^i \otimes \left(\mathrm{FM}_{\mathbf{K}}(\vec{a} \otimes \mathbf{w}^i)\right)$
    3. $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} \otimes (\mathbf{d}^i)^{\boldsymbol{\alpha}_i}$
  for $i = 1, \ldots, k$
    4. $\mathbf{v}^i \leftarrow \mathbf{v}^i \otimes \boldsymbol{\mu} \oslash \mathbf{d}^i$
return $\boldsymbol{\mu}$

D.1.2. Details on Baselines

For the BF in separation integration, we first compute the pairwise *shortest path distances* for all vertices, using vertices and edges in the input mesh. We then compute the *element-wise exponential* $\mathbf{K}$ with $\mathbf{K}_{ij} := \exp(-\lambda \mathrm{dist}(i,j))$ for all i, j.

For the BF in diffusion integration, we use the vertex embeddings of the input mesh and create a graph G with edges between nodes i and j if $\|\mathbf{n}_i - \mathbf{n}_j\|_1 \leq \epsilon$. After creating the set of edges, we compute the *matrix exponential* $\mathbf{K} = \exp(\lambda \mathbf{W}_{\mathrm{G}})$ with $(\mathbf{W}_{\mathrm{G}})_{ij} := \|\mathbf{n}_i - \mathbf{n}_j\|_1 \cdot \mathbb{1}[\|\mathbf{n}_i - \mathbf{n}_j\|_1 \leq \epsilon]$.

In steps 1 and 2 in the algorithm 1, both the baseline variants explicitly perform $\mathbf{Kx}$, the matrix-vector multiplication.

D.1.3. Details on Hyper-parameters

For diffusion integration, we fix parameters $\epsilon = 0.01$ and $\lambda = 0.5$, and the number of random features is 30. For separation integration, we choose $\lambda = 0.2$, unit-size $= 0.1$, threshold $= 2000$ (the maximum size of the graph, measured in the number of vertices, for which the integrator is conducted in a brute-force manner).

For computations of the Wasserstein barycenters, the input vector $\vec{a}$ contains area weights for vertices. The area weights are proportional to the sum of triangle areas adjacent to each vertex in a triangle mesh (Solomon et al., 2015). We set the

number of input distributions $k = 3$ and $\boldsymbol{\alpha} = \frac{\vec{1}}{k} = \frac{\vec{1}}{3}$. For each mesh, we generate three different input distributions $\boldsymbol{\mu}^i$, each with mass concentrated in vertices surrounding a distinct center vertex.

The experiments performed for Wasserstein barycenter are conducted on an 8-CPU core Ubuntu virtual machine on Google Cloud Compute.

D.1.4. ADDITIONAL EXPERIMENTS

In Table 5, we include comparisons of RFD with an additional baseline (Solomon et al., 2015). Even though there are similarities between our work and the above mentioned autors, we would like to point the following key differences : (1) Solomon et al. (2015) does not consider an ϵ-graph and (2) uses the heat kernel which is constructed using mesh Laplacian instead of our matrix exponential of the weighted adjacency matrix.

Moreover, we note that the kernel employed in our SF experiments can be seen as a generalization of the Laplace kernel on the manifold, and thus not directly comparable to the heat kernel.

Table 5. Comparison of the total runtime and mean-squared error (MSE) across several meshes for diffusion-based integration. Slmn is the integrator from (Solomon et al., 2015). Runtimes are reported in seconds. The lowest time for each mesh is shown in bold. MSE is calculated w.r.t. the output of brute force (BF).

Mesh	$\lvert V\rvert$	**Total Runtime**			**MSE**	
		BF	Slmn	RFD	Slmn	RFD
Alien	5212	8.06	0.57	**0.39**	0.042	0.041
Duck	9862	45.36	1.94	**1.10**	0.002	0.002
Land	14738	147.64	4.17	**2.17**	0.023	0.017
Octocat	18944	302.84	6.74	**3.36**	0.022	0.027

D.2. Gromov Wasserstein Distance

The optimal transport (OT) problem associated with Gromov-Wasserstein (*GW*) discrepancy (Peyré et al., 2016), which extends the Gromov-Wasserstein distance (Mémoli, 2011), has emerged as an effective transportation distance between structured data, alleviating the incomparability issue between different structures by aligning the intra-relational geometries. The *GW* discrepancy problem can be solved iteratively by conditional gradient method (Peyré et al., 2016) and the proximal point algorithm (Xu et al., 2019). *GW* distance is isometric, meaning the unchanged similarity under rotation, translation, and permutation and is thus related to graph matching problem, encoding structural information to compare graphs, and has also been successfully adopted in image recognition (Peyré et al., 2016), alignment of large single-cell datasets (Demetci et al., 2020), and point-cloud data alignment (Mémoli & Sapiro, 2006). However, despite its broad use, the Gromov Wasserstein distance is computationally expensive as it scales as $O(n^2m^2)$ where n, m are the numbers of source and target nodes respectively.

D.2.1. (FUSED) GROMOV WASSERSTEIN DISCREPANCY

Formally, the Gromov-Wasserstein discrepancy between two measured similarity matrices $(\mathbf{C}, \mathbf{p}) \in \mathbb{R}^{n\times n} \times \Sigma_n$ and $(\mathbf{D}, \mathbf{q}) \in \mathbb{R}^{m\times m} \times \Sigma_m$ is defined as :

$$\begin{aligned} \mathrm{GW}(\mathbf{C}, \mathbf{D}, \mathbf{p}, \mathbf{q}) &= \min_{\mathbf{T}\in\mathcal{C}_{\mathbf{p},\mathbf{q}}} \sum_{i,j,k,l} \ell(\mathbf{C}_{i,k}, \mathbf{D}_{j,l})\mathbf{T}_{i,j}\mathbf{T}_{k,l} \\ &= \min_{\mathbf{T}\in\mathcal{C}_{\mathbf{p},\mathbf{q}}} \langle L(\mathbf{C}, \mathbf{D}, \mathbf{T}), \mathbf{T}\rangle \end{aligned} \tag{38}$$

where $\mathbf{C}$ and $\mathbf{D}$ are matrices representing either similarities or distances between nodes within the graph, $\mathbf{A}_{i,j}$ is the ijth entry of the matrix $\mathbf{A}$, ℓ is the loss function applied elementwise on the matrices. The common choices of the loss function are Euclidean distance, i.e. $\ell(x, y) := (x - y)^2$ or KL-divergence, i.e. $\ell(x, y) := x \log \frac{x}{y} - x + y$, $\mathbf{p} \in \mathbb{R}^n_+$ (resp. $\mathbf{q} \in \mathbb{R}^m_+$), $\sum p_i = 1$ (resp. $\sum q_i = 1$), is the probability simplex of histograms with n (resp. m) bins, and $\mathbf{T}$ is the coupling matrix

between the two spaces on which the similarity matrices are defined, i.e.

$$\mathcal{C}_{\mathbf{p},\mathbf{q}} = \{\mathbf{T} \in \mathbb{R}_+^{n\times m} \mid \mathbf{T}\mathbf{1}_m = \mathbf{p}, \mathbf{T}^\top \mathbf{1}_n = \mathbf{q}\} \tag{39}$$

Define, $L(\mathbf{C}, \mathbf{D}, \mathbf{T}) := [\mathbf{L}_{k,l}] \in \mathbb{R}^{n\times m}$ and $\mathbf{L}_{k,l} := \sum_{i,j} \ell(\mathbf{C}_{i,k}, \mathbf{D}_{j,l})\mathbf{T}_{i,j}$ and $\langle \cdot, \cdot \rangle$ is the inner product of matrices. For the rest of the section, we use the Euclidean distance ℓ as our loss function.

Following the work of (Vayer et al., 2018), the concept of *GW* discrepancy can be extended to an OT discrepancy on graphs called Fused Gromov Wasserstein (FGW) that take into account both the node features of the graphs as well as their structure matrices. *FGW* can be written as follows:

$$\begin{aligned}\mathrm{FGW}_\alpha(\mathbf{C}, \mathbf{D}, \mathbf{M}, \mathbf{p}, \mathbf{q}) = \min_{\mathbf{T}\in\mathcal{C}_{\mathbf{p},\mathbf{q}}} \sum_{i,j,k,l} ((1-\alpha)\mathbf{M} + \\ \alpha\ell(\mathbf{C}_{i,k}, \mathbf{D}_{j,l}))\mathbf{T}_{i,j}\mathbf{T}_{k,l}\end{aligned} \tag{40}$$

where $\mathbf{M}$ is the distance matrix encoding differences between the nodes of the 2 graphs and α is the convex combination between the distance matrices.

D.2.2. Estimating the Action of Hadamard Square of Matrices on Vectors

To compute (Fused) Gromov-Wasserstein discrepancies, one needs to compute $\mathbf{C}^{\odot 2}\mathbf{p}$, where $\mathbf{C}^{\odot 2}$ is the element-wise square (Hadamard square) of a cost matrix $\mathbf{C}$ and a vector $\mathbf{p}$. This is given by the following formula :

$$\mathbf{C}^{\odot 2}\mathbf{p} = \mathrm{diag}\left(\mathbf{C}\mathbf{D}_\mathbf{p}\mathbf{C}^\top\right) \tag{41}$$

where diag($\mathbf{M}$) is the diagonal of $\mathbf{M}$ and $\mathbf{D}_\mathbf{p}$ is the matrix formed by $\mathbf{p}$ as its diagonal.

However, for all our fast variants, we never materialize the matrix $\mathbf{C}$ explicitly. Thus to estimate the above action, we can make 2 calls to our Fast Multiplication method (FM) via the following :

$$\mathbf{C}^{\odot 2}\mathbf{p} \sim \mathrm{diag}(FM_\mathbf{C}(FM_\mathbf{C}(\mathbf{D}_\mathbf{p})^\top)) \tag{42}$$

Here FM can either be SeparationFactorization (SF) or the RFDiffusion algorithm (RFD), and for clarity, we use the subscript for the matrix $\mathbf{C}$ to specify that we are approximating the (right) action of the matrix $\mathbf{C}$.

D.2.3. Algorithm to Put It All Together

To calculate the OT (for Gromov-Wasserstein and Fused Gromov Wasserstein), the loss matrix $L(\mathbf{C}, \mathbf{D}, \mathbf{T})$ needs to be computed, which is one of the most expensive steps, as it involves a tensor-matrix multiplication. Indeed if the source graph has n nodes and the target graph has m nodes, this operation has a time complexity of $O(n^2m^2)$. However, when the loss function ℓ can be written as $\ell(a, b) = f_1(a) + f_2(b) - h_1(a)h_2(b)$ for functions (f_1, f_2, h_1, h_2), the loss matrix can be calculated as (Peyré et al., 2016)

$$L(\mathbf{C}, \mathbf{D}, \mathbf{T}) = f_1(\mathbf{C})\mathbf{p}\mathbf{1}_m^\top + \mathbf{1}_n\mathbf{q}^\top f_2(\mathbf{D}) - h_1(\mathbf{C})\mathbf{T}h_2(\mathbf{D})^\top \tag{43}$$

where the functions (f_1, f_2, h_1, h_2) are applied elementwise. In this case, the time complexity reduces to $O(n^2m + m^2n)$. Moreover if ℓ is the Euclidean loss function, then $f_1(x) = f_2(x) = x^2$ and $h_1(x) = x, h_2(x) = 2x$.

Our Fast Multiplication methods (FM) can be used to efficiently estimate the above tensor product given by equation 43 via the algorithm 2 thus leading to computation gains in computing *GW* (resp. *FGW*) discrepancies. In the above algorithm, the implicit representation of a matrix $\mathbf{M}$ can be given as an array of 3-D coordinates and hyperparameters that are specific to the chosen FM algorithm.

Our contributions go further than providing fast accurate computation of the tensor products but also a fast computation of the line search algorithm (Algorithm 2 as presented in (Titouan et al., 2019)). The line search algorithm provides an optimal step size for the *FGW* iterations.

We now provide a brief description of how our novel FM methods can be injected into the line search algorithm for the conjugate gradient. The line search algorithm at a *FGW* iteration takes in the structure matrices of the source and target graphs (which in our case will be implicit representations of such matrices), transport cost $\mathbf{G}$, $d\mathbf{G}$ which is the difference

Algorithm 2 Fast Computation of Tensor Products

Input: $\mathbf{T}$ and $I_{\mathbf{C}}, I_{\mathbf{D}}$ $\{I_{\mathbf{M}} :=$ implicit representation of the cost matrix $\mathbf{M}\}$
Output: $L(I_{\mathbf{C}}, I_{\mathbf{D}}, \mathbf{T})$
1. Estimate $\mathbf{v}_1 := f_1(\mathbf{C})\mathbf{p}$ by Equation 42
    Compute $\mathbf{w}_1 = \mathbf{v}_1\mathbf{1}^\top$
2. Estimate $\mathbf{v}_2 := f_2(\mathbf{D})\mathbf{q}$ by Equation 42 {Using the fact that $\mathbf{D}$ is symmetric}
    Compute $\mathbf{w}_2 = \mathbf{1}\mathbf{v}_2^\top$
3. Estimate $h_1(\mathbf{C})\mathbf{T}h_2(\mathbf{D})^\top$ by
    $\mathbf{w}_3 := (FM_{\mathbf{D}}(FM_{\mathbf{C}}(\mathbf{T})^\top))^\top$
return $\mathbf{w}_1 + \mathbf{w}_2 - 2\mathbf{w}_3$

Algorithm 3 Fast Computation of Line-search for CG

Input: $I_{\mathbf{C}}, I_{\mathbf{D}}, \alpha, \mathbf{G}, d\mathbf{G}, \mathbf{M}$,
Output: Optimal Step Size τ
Estimate $c_{\mathbf{C},\mathbf{D}}$ by Step 1 and 2 of algorithm 2.
Estimate $a_1 := \mathbf{C}d\mathbf{G}\mathbf{D}$ by $FM_{\mathbf{D}}(FM_{\mathbf{C}}(d\mathbf{G})^\top)^\top$ {since $\mathbf{D}$ is symmetric}.
Compute $a := -2\alpha\langle a_1, d\mathbf{G}\rangle$.
Estimate $b_1 := \mathbf{C}\mathbf{G}\mathbf{D}$ by $FM_{\mathbf{D}}(FM_{\mathbf{C}}(\mathbf{G})^\top)^\top$
Compute $b := \langle(1-\alpha)\mathbf{M} + \alpha c_{\mathbf{C},\mathbf{D}}, d\mathbf{G}\rangle - 2\alpha(\langle a_1, \mathbf{G}\rangle + \langle b_1, d\mathbf{G}\rangle)$
Compute $c :=$ cost$(\mathbf{G})$
if $a > 0$ **then**
    $\tau \leftarrow \min(1, \max(0, \frac{-b}{2a}))$
else
    if $a + b < 0$ **then**
        $\tau \leftarrow 1$
    else
        $\tau \leftarrow 0$
    end if
end if

between the optimal map found by linearization in the *FGW* algorithm and $\mathbf{G}$, and $\mathbf{M}$, a matrix measuring the differences between nodes features of source and target graphs. Define $c_{\mathbf{C},\mathbf{D}} := f_1(\mathbf{C})\mathbf{p}\mathbf{1}_m^\top + \mathbf{1}_n\mathbf{q}^\top f_2(\mathbf{D})$. Finally, the algorithm needs a cost function that combines the transportation cost coming from the node features and the graph structures which is applied to $\mathbf{G}$. This cost function crucially relies on the tensor product computation (Equation 43) and our Algorithm 2 provides a fast efficient computation of this cost function as well.

Note that employing a low-rank decomposition of the cost matrices to speed up the computation of *GW* has also been studied in (Scetbon et al., 2021). However, our work differs from their work in certain key aspects. The choice of our kernel matrices and the method of factorization of the cost matrix differs from the above work. Moreover, we do not design our methods with *GW* computations in mind but a flexible mechanism that can be injected into various *GW* computations including entropic-*GW* (similar to Algorithm 2 proposed in (Scetbon et al., 2021)).

D.2.4. GROMOV WASSERSTEIN BARYCENTERS

Recall, that given graphs $G_1, \cdots, G_N$, where $G_i := \{\mathbf{C}_i, \mathbf{p}_i\}$ comes equipped with a cost matrix $\mathbf{C}_i$ between its nodes and a probability simplex defined on its nodes, the Wasserstein barycenter can be defined as the minimizer of the functional

$$F[\nu] = \sum_{i=1}^{n} w_i \, \mathrm{GW}((\bar{\mathbf{C}}, \mathbf{C}_i, \bar{\mathbf{p}}, \bar{\mathbf{p}_i}) \tag{44}$$

where w_i are some fixed positive weights and $\sum w_i = 1$, $\bar{G} := \{\bar{\mathbf{C}}, \bar{\mathbf{p}}\}$ is the predefined barycenter graph with a fixed number of nodes. One can similarly define a Fused *GW* barycenter as well.

As an application of our methods, we interpolate between a bunny (1887 vertices) and a torus (1949 vertices). We center

the meshes around $(0, 0, 0)$ and scale the coordinates such that $|x|, |y|, |z| \leq 1$. We then run a fast Sinkhorn barycenter algorithm (Janati et al., 2020) to get a configuration of intermediate shapes in 3D space. A sampling algorithm (Voxel Grid Filter) is then used to reduce the density of the generated point clouds to 1445, 1450, and 1425 points respectively. We then try to solve for the edges of these intermediate point clouds. Our method as well as the baseline *GW*-cg algorithm produce

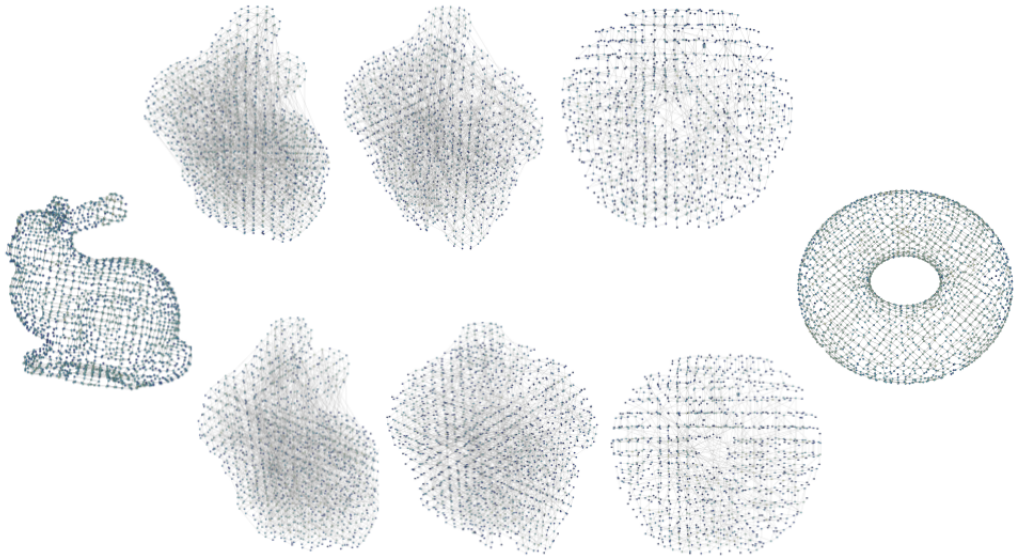

Figure 8. Interpolation between a bunny and a torus. Barycenters are computed using, Top row: *GW*-cg, Bottom row : *GW*-cg-RFD

decent meshes and tries to preserve the consistency of the manifold mesh throughout the interpolation (Figure 8). For the barycenter experiment, we use $m = 16$ random features, $\lambda = -.15$, and $\epsilon = .13$.

All experiments on *GW* and its variants are conducted on a Google Colab.

E. Ablation Studies

In this section, we present detailed ablation studies for our experiments.

E.1. Ablation Studies for Vertex Normal Prediction Experiments

RFDiffusion. There are three hyper-parameters in our RFDiffusion algorithm, which are the number of random features m, epsilon ϵ, and lambda λ. The number of random features determines the accuracy of our approximation of the weighted adjacency matrix $\mathbf{W}_{\mathrm{G}}$. The epsilon hyper-parameter controls the sparsity level of the ϵ-NN graph. The lambda hyper-parameter controls the "steepness" of our kernel. We can make the following conclusions based on the observations from Fig. 9. Firstly, increasing the number of random features usually gives us a better estimation of the weighted adjacency matrix $\mathbf{W}_{\mathrm{G}}$, which leads to higher cosine similarity. Secondly, a densely connected graph (large epsilon) coupled with a steeper kernel function (lambda with large absolute value) leads to better performance.

SF. There are two hyper-parameters in our SF algorithm, which are: $\mathrm{unit\text{-}size}$ (determining the quantization mechanism: all the shortest path lengths are considered modulo $\mathrm{unit\text{-}size}$) and $\mathrm{threshold}$ (specifying the maximum size of the graph, measured in the number of vertices, for which the GFI is conducted in a brute-force manner). Fig. 10 shows pre-processing time, interpolation time, and cosine similarity under different values of the unit-size hyper-parameter. The results are reported with the threshold set as half of the number of vertices in the mesh. We can observe from the plots that a small value for unit-size provides a better estimation of the shortest-path distance without incurring significant changes in pre-processing and interpolation time. Fig. 11 shows the ablation of different thresholds while keeping the unit-size hyper-parameter the same (0.01). There is a trade-off between accuracy (measured by cosine similarity) and interpolation time. In the main body of our paper, we set the unit-size to 0.01 and the threshold to 0.5.

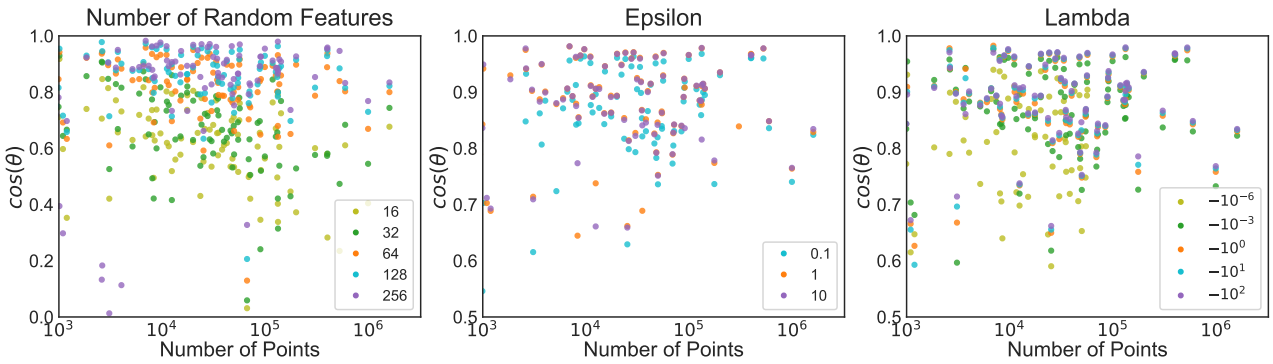

Figure 9. Ablations study for RFDs on the vertex normal prediction task.

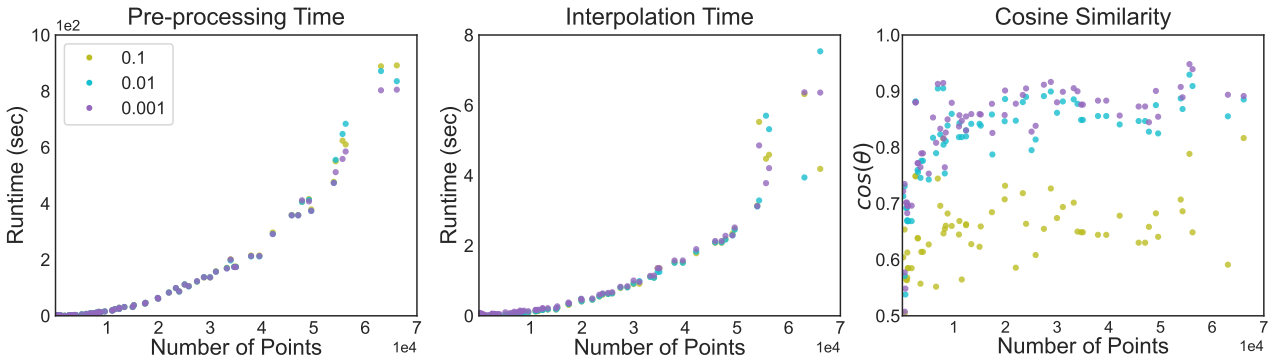

Figure 10. Ablation study for the **unit-size** hyper-parameter in SF algorithm for vertex normal prediction task.

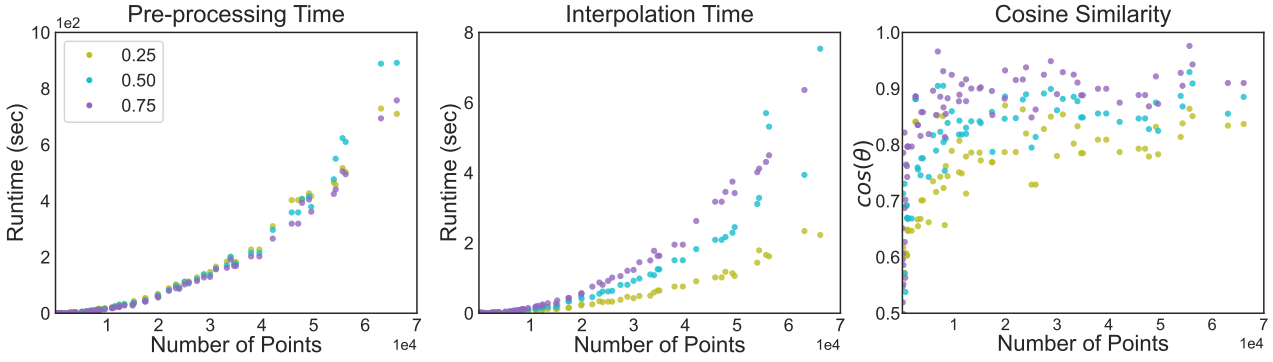

Figure 11. Ablation study for the **threshold** hyper-parameter in SF algorithm for vertex normal prediction task.

E.2. Ablation Studies for Gromov Wasserstein experiments

The ϵ parameter in our RFDiffusion effectively controls the sparsity of our source and the target graphs. We find that our runtimes for *GW* distance via the conditional gradient algorithm remain mostly stable while that of the baseline algorithm grows with the density of the graph. However, runtime for *GW*-distance computed via the proximal point algorithm is fairly stable (Xu et al., 2019). Surprisingly, the runtime for the *FGW* distance is also stable. We hypothesize that it is because even though the source or target graphs are sparse, we need to materialize a dense cross-feature distance matrix between the node features of the source and target nodes. In these cases, the runtimes for our RFDiffusion-integrated *GW* (with conditional gradient algorithm) and *FGW* are also stable and inconsistently lower than the baseline methods. The middle figure 12 shows the relative error as the function of ϵ. As ϵ increases, for a fixed value of λ, the action of the matrix that we are trying to estimate will have a larger norm, and thus the relative error grows in accordance to Lemma 2.6. However, for meshes rescaled in a unit box, the ϵ tends to be smaller in practice.

We also see similar behavior with λ, i.e., smaller values of $|\lambda|$ tend to produce better results. This phenomenon is predicted

by Lemma 2.6. However, if λ gets too close to 0, the structure matrices approach an identity matrix, leading to information loss. This causes instabilities in the convergence of the algorithm.

All the experiments are run on random 3D distributions with 3000 points, and the results are averaged over 10 runs.

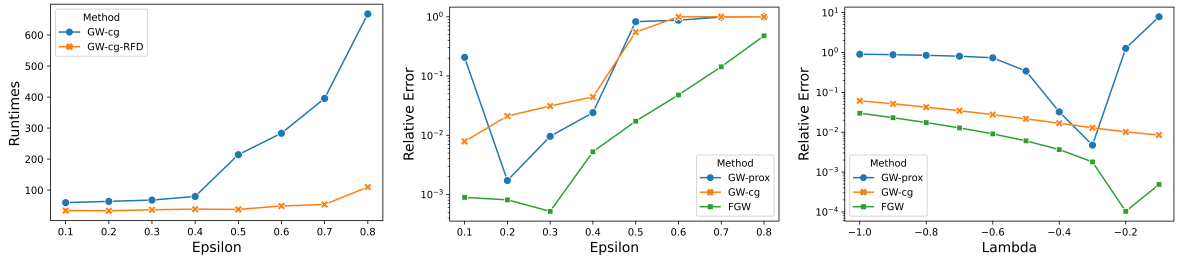

Figure 12. Ablation study over the λ and ϵ parameters for the *GW* variants. **Left:** The runtime for the baseline *GW*-cg method increases as the input graph gets denser while our runtimes remain mostly constant. **Middle and right:** Plots show relative error as a function of ϵ and λ respectively. The relative error increases if the graphs get "dense" or the kernel becomes too "steep."

E.3. Ablation Studies on Wasserstein Barycenter Experiments

In Table 7, we provide ablation results for the λ hyper-parameter in RFD algorithm for the Wasserstein barycenter task. Experiments are conducted on the mesh `duck`. We show that the MSE increases with λ, which is in line with the observation in Section E.2. The runtime is nearly unchanged for different values of λ. We normalize the coordinates of the vertices and choose the epsilon parameter to be 0.01, making the computation meaningful. Larger epsilon values will cause the graph to be too dense, and smaller epsilon values will create an epsilon graph with almost no edges.

In Table 6, we provide ablation results for the **unit-size** hyperparameter in the SF algorithm. We show that the MSE slowly increases with unit-size, and the runtime is nearly unchanged for different values of **unit-size**.

Table 6. Ablation study for the **unit-size** parameter in SF for Wasserstein barycenter task.

unit-size	**MSE**	**Total time (secs)**
0.1	2.1×10^{-3}	19.4
0.5	2.1×10^{-3}	19.1
1.0	2.1×10^{-3}	18.9
5.0	2.7×10^{-3}	18.8
10.0	3.1×10^{-3}	19.1

Table 7. Ablation study for the λ parameter in RFD for Wasserstein barycenter task.

λ	**MSE**	**Total time (secs)**
0.1	2×10^{-4}	1.1
0.3	1.1×10^{-3}	1.1
0.5	2.1×10^{-3}	1.0
0.7	2.7×10^{-3}	1.1
0.9	3.3×10^{-3}	1.1

F. Graph Classification Experiments using the RFD Kernel

We extract our RFD kernel and use it for various graph classification tasks. More specifically, we compute the top k eigenvalues for the approximated kenel matrix and pass it to a random forest classifier for classification. Note that, as described in (Nakatsukasa, 2019), low-rank decomposition of the kernel matrix (provided directly by the RFDiffusion method via the random feature map mechanism) can be used to compute efficiently eigenvectors and the corresponding eigenvalues.

However, most of the benchmark datasets for graph classification are molecular datasets (Morris et al., 2020). Our methods are originally developed for meshes and point clouds where we excel (see section 3.3) hence we do not consider molecular graphs in the main paper. The node features of these molecular graphs are extremely coarse and thus the epsilon-neighborhood graph constructed using these features performs poorly in downstream graph classification tasks. We apply our RFDiffusion kernel on the sets of points, considering the node features as vectors in a d-dimensional space. The RFD kernel produces a smoothened version of the epsilon-neighborhood graph, giving good results even when the baseline applying explicitly the epsilon-neighborhood graph does not. This is the case since random features replace the

Table 8. Graph Classification using RFD Kernel

Dataset	# Graphs	Avg. # Nodes	Avg. # Edges	VH	RW	WL-SP	FB	RFD (ours)
MUTAG	188	17.93	19.79	69.1	81.4	81.4	**84.7**	71.0
ENZYMES	600	32.63	62.14	20.0	16.7	27.3	**29.0**	27.0
PROTEINS	1113	39.06	72.82	71.1	69.5	72.1	70.0	**75.0**
NCI1	4110	29.87	32.3	55.7	TIMEOUT	60.8	**62.9**	61.0
DD	1178	284.32	715.66	74.8	OOM	**76.0**	-	73.0
PTC-MR	344	14.29	14.69	57.1	54.4	54.5	55.6	**61.0**

combinatorial object (a graph with edges and no-edges) with its "fuzzy" version, where all the nodes are connected by edges (that are not explicitly reconstructed though) but the weights corresponding to non-edges in the original graph are close to zero with high probability.

We compare our algorithm with four baselines : Vertex Histogram (VH), Random Walk (RW), Weisfeiler-Lehman shortest path kernel (WL-SP) (Nikolentzos et al., 2022) and Feature based method (FB) (de Lara & Pineau, 2018). Our method compares favorably with these methods and is also competitive with various kernel methods reported in (de Lara & Pineau, 2018; Balcilar et al., 2020; Nikolentzos et al., 2022; Seenappa et al., 2019). The results along with statistics about the datasets are summarized in Table 8.