A Domain-Oblivious Approach for Learning Concise Representations of Filtered Topological Spaces

Yu Qin, Brittany Terese Fasy, Carola Wenk, and Brian Summa

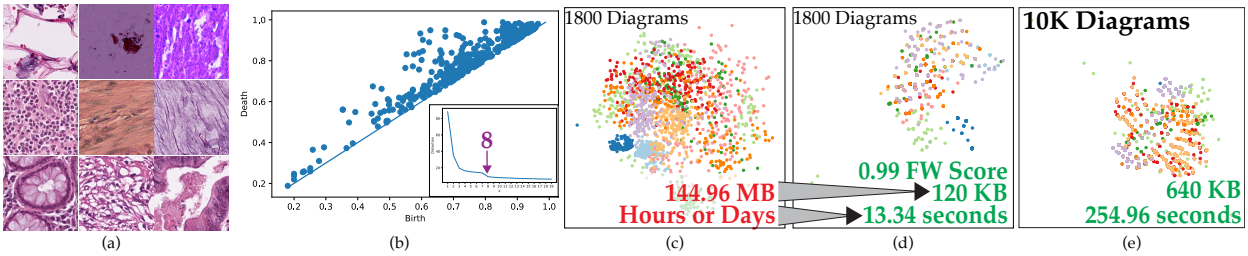

Fig. 1: (a) Examples from a dataset of 10k histology images of colorectal cancer. (b) An example persistence diagram that encodes the topological structure in an image. Inset illustrates an elbow method plot run from clustering a subset of 1800 images using 1-Wasserstein distance on each image's diagram. This shows there are approximately 8 topologically distinct clusters. (c) Clustering result using 1-Wasserstein distances on the subset. (d) Our high-quality concise representation uses only a fraction of the memory and computation time. (e) Our approach scales the full dataset, which is not feasible with 1-Wasserstein.

Abstract—Persistence diagrams have been widely used to quantify the underlying features of filtered topological spaces in data visualization. In many applications, computing distances between diagrams is essential; however, computing these distances has been challenging due to the computational cost. In this paper, we propose a persistence diagram hashing framework that learns a binary code representation of persistence diagrams, which allows for fast computation of distances. This framework is built upon a generative adversarial network (GAN) with a diagram distance loss function to steer the learning process. Instead of attempting to transform diagrams into vectorized representations, we hash diagrams into binary codes, which have natural advantages in large-scale tasks. The training of this model is domain-oblivious in that it can be computed purely from synthetic, randomly created diagrams. As a consequence, our proposed method is directly applicable to various datasets without the need of retraining the model. These binary codes, when compared using fast Hamming distance, better maintain topological similarity properties between datasets than other vectorized representations. To evaluate this method, we apply our framework to the problem of diagram clustering and we compare the quality and performance of our approach to the state-of-the-art. In addition, we show the scalability of our approach on a dataset with 10k persistence diagrams, which is not possible with current techniques. Moreover, our experimental results demonstrate that our method is significantly faster with less memory usage, while retaining comparable or better quality comparisons.

Index Terms—Topological data analysis, Persistence diagrams, Persistence diagram distances, Learned hashing, Clustering.

1 Introduction

The features quantified by topological data analysis (TDA) [24] have been shown to express the fundamental structure of scalar fields in a way that is generally applicable to many domains. As such, TDA approaches such as persistent homology [25], contour trees [14], Reeb graphs [8, 54], and Morse(-Smale) complexes [22, 34] can be found to encode meaningful structure in a variety of research applications such as 3D shape matching [16], combustion physics [10,33], nuclear physics [50], fluid dynamics [38], chemistry [7,32], Alzheimer's disease [48], autism spectrum disorders [47,62], cancer histology [45], protein folding [81], and bio-molecular analysis [52]. There has also been recent work in using TDA quantification as input to machine learning [15,45,57,58].

As described in detail in Section 2.1, a persistence diagram is the most common way to present and encode the topological structure in a dataset. As such, many TDA approaches compute the distance between these diagrams as way to express the topological (dis)similarity between datasets. For instance, for approaches that compute clusters based on topological similarly [16,45,73,76], computing the distance between diagrams is a fundamental operation.

Although, computing distances is costly. Consider that the most widely accepted diagram distance measures, Wasserstein distances, require a expensive matching of all points between two diagrams. As discussed in Section 2, to combat this complexity, there have been many approaches to try to reduce this cost. The goal of our work is the same, but provides significant advances over the state-of-the-art. As we detail in this paper, we provide a new representation to express topological structure that is more concise than previous works, but also leads directly to faster calculations of distances. In particular, we show how to reduce diagrams to simple binary codes (64-bit integers). The key to this representation is a learned hash function. As we show, this hash can be learned purely from random, synthetically generated diagrams. We have found that the only constraint on generating training data is that the generated diagrams should have approximately the same average number of diagram points as are in the test data. This means that training is domain-oblivious with a model being potentially used on a wide variety of datasets without the need for retraining. In this new representation, distance calculation is simply a bit-wise count comparison (Hamming distance) of binary codes. This gives a distance calculation that is extremely fast and scalable. We illustrate this scaling through the clustering of a dataset with 10k diagrams, a size which is not achievable for several approaches.

- *Yu Qin is with Tulane University. E-mail: yqin2@tulane.edu.*
- *Brittany Terese Fasy is with Montana State. E-mail: brittany.fasy@montana.edu.*
- *Carola Wenk is with Tulane University. E-mail: cwenk@tulane.edu.*
- *Brian Summa is with Tulane University. E-mail: bsumma@tulane.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

1.1 Contributions

The specific contributions of this work are:

- A concise binary code representation of persistence diagrams that maintains topological (dis)similarity in Hamming space;
- A procedure to train the binary code hash function that can run purely on synthetic data and therefore is domain-oblivious; and
- Applications to topological clustering of real-world datasets that provide: Significant comparison speedups, lower memory footprints, and comparable or better quality clustering results than other vectorized representations of persistence diagrams.

2 Background and Related Work

This section outlines the technical background for persistent homology and hashing, as relevant to the methods developed in this paper.

2.1 Persistent Homology and Persistence Diagrams

Given a dataset $\mathbb{X}$, we describe the persistence homology as a nested sequence of spaces (called a *filtration*) by computing the homology of the filtration. We can think of the filtration parameters (usually indexed by $\mathbb{R}$ or $\mathbb{Z}$) as *scales*. The result is a summary of the homology (or, connectivity and "higher-order" connectivity) throughout the different scales of the filtration; colloquially, the homology captures the connected components, holes or loops, and voids found in the data, and the persistent homology tracks them through scales. Examples of such filtrations are: nested sublevel sets of a height function when $\mathbb{X}$ is embedded in Euclidean space [74], and a sequence of Vietoris-Rips complexes using increasing radius when $\mathbb{X}$ is sampled from a metric space [35, 77]. The homology of this filtration is captured in a finite set of *birth-death* pairs; that is, each homology feature has a "birth" scale, where the feature first appears, and a "death" scale, where that feature merges with a feature existing before it. The *persistence diagram* is a multiset of points in the extended plane $\overline{\mathbb{R}}^2$, where each homology feature is represented as a point whose x-coordinate is its birth scale and whose y-coordinate is its death scale. Here, the extended real line $\overline{\mathbb{R}}$ is the real line $\mathbb{R}$ plus $\pm\infty$, which allows for us to represent homology features that are born but never die (the *essential classes* of the filtration). See Fig. 1(b) for an example diagram used in this work.

Wasserstein and Bottleneck Distances Letting $\mathscr{D}$ denote the collection of all diagrams, we can equip it with structure by defining distances between diagrams. For example, for $q \in \mathbb{N}$, the q-Wasserstein distance $d_q \colon \mathscr{D} \times \mathscr{D} \to \mathbb{R}$ is defined by

$$d_q(p_1, p_2) := \min_M \left(\sum_{(a,b)\in M} ||a-b||_\infty^q + \sum_{a \in M^c} |a_x - a_y|^q \right)^{1/q},$$

where M ranges over all matchings between persistence diagrams p_1 and p_2, and M^c is the set of points in $p_1 \sqcup p_2$ that do not appear in the matching M; see [20, 37, 40, 53]. Letting $q \to \infty$, we obtain what is known as the bottleneck distance (or, equivalently, the interleaving distance) [19]. For this work, we use the one-Wasserstein (W1) in our experiments, as this is a popular choice in applications [1, 15]. These diagrams are Lipschitz stable in the presence of slight perturbations (noise) in data [20]. Given this stability, diagrams that are close in distance are often considered be topologically similar. Computationally, however, both the Wasserstein and the bottleneck distances are expensive, as they require computing the optimal matching between points in the two diagrams. In particular, computing the Wasserstein distance between two diagrams costs $O(n^2 \log^2 n)$ time, where n is the total number of simplices (and can be exponential in the size of $\mathbb{X}$) [75].

Approximating Distances When many distances between diagrams need to be computed, the roughly quadratic calculation can be daunting. Thus, several approaches have been introduced to approximate computing the Wasserstein distances [2, 6, 40, 61, 63]. One approach that is quite successful is to simplify the input representation, before a persistence diagram is even computed [36, 78]. However, this makes assumptions on the underlying domain (e.g., it must be a 2D or 3D image). Working with the persistence diagram itself, we can turn to treating the set of points in the diagram as a geometric set.

Kerber et al. [40] introduced an approximate Wasserstein distance algorithm to accelerate the computation of the matching using a k-d tree. This iterative Wasserstein (IW) computation allows a bound for either quality or time as input to the approximation algorithm in order to introduce a halting condition for the iterative algorithm. This algorithm was extended by Vidal et al. [76] for the problem of computing barycenters of persistence diagrams. Although fast, the memory requirements for this approach are substantial, as we illustrate in Section 4.4.

Let $M \in \mathbb{R}$ and let $\mathscr{D}_M$ denote the subspace of $\mathscr{D}$ that fits inside the box of radius M centered at the origin; that is:

$$\mathscr{D}_M = \{p \in \mathscr{D} \mid ||p||_\infty \leq M\}.$$

Then, for an input parameter $d \in \mathbb{N}$, we can create a $d \times d$ grid G over $[-M, M]^2$. We define a histogram, where we count the number of persistence points in each grid cell. This approach is used in [26, 44]. Snapping the points in the persistence diagrams to the center of the nearest histogram bin, [26] uses the snap-rounded diagrams to design a data structure that supports searching and [44] computes the Wasserstein distance between the snap-rounded diagrams, leveraging the fact that the coordinates come from a finite set. While this work can benefit from several fast optimal transport computation approaches [21, 64], it still poses significant costs for distance calculations. In our experimental section, we call the Wasserstein distance between snap-rounded diagrams the Histogram Wasserstein (HW) distance.

2.2 Other Topological Descriptors

Leveraging histograms of the persistence points, or snap-rounded persistence diagrams, can be extended to considering the distributions of the persistence points. In fact, density estimators built over the persistence points have been studied by several groups of researchers independently [1, 4, 18, 57, 58]. In addition, kernel-based methods [15, 43, 46] and functional summaries of persistence diagrams [5, 11, 59] allow the use of functional data analysis, which has been studied for decades. One benefit of considering functional summaries such as the density estimators is that they can be vectorized, which leads to studying functional and vectorized summaries of persistence diagrams. With vectorized representations, the space cost of the discretization can be weighed against the speedup gained by computing L_p distances between vectors instead of distances between persistence diagrams.

Persistent Images (PI) The persistence image (PI) is a discretization of a weighted kernel density estimator (a non-parametric density estimator) built on the rotated points of a persistence diagram [1]). Like the histogram, this discretization depends on a bounding box around the off-diagonal points in the persistence diagram. In practice, using the PI requires choosing: a bounding box, a discretization resolution, and a weight function on the set of points in the persistence diagram. Choosing these parameters can be non-trivial in practice, but various heuristics have been successfully employed [1, 18, 83].

Betti Curves (BC) Tracking the homology as the scale increases in the filtration might be more information than needed for many applications. Other topological invariants that can be tracked include the Euler characteristic and the Betti numbers. For a given integer $k \in \mathbb{N}$, the k^{th} Betti curve is the rank of the homology group of the filtered space as the scale parameter increases. For a fixed dimension d, the *Betti curves (BC)* are a piece-wise constant function $BC \colon \mathbb{R} \to \mathbb{Z}^d$, where the k^{th} coordinate of BC is the k^{th} Betti curve. The use of the Betti curves dates back to Robins in the early 2000s (see [59]), but has also been used in [28, 82]. The L_p-distance between Betti curves can be computed explicitly, or can be approximated by sampling the domain. Often, the latter approach is preferred in practice. Thus, as with PI, using the Betti curves requires selecting a bounding box and a discretization resolution.

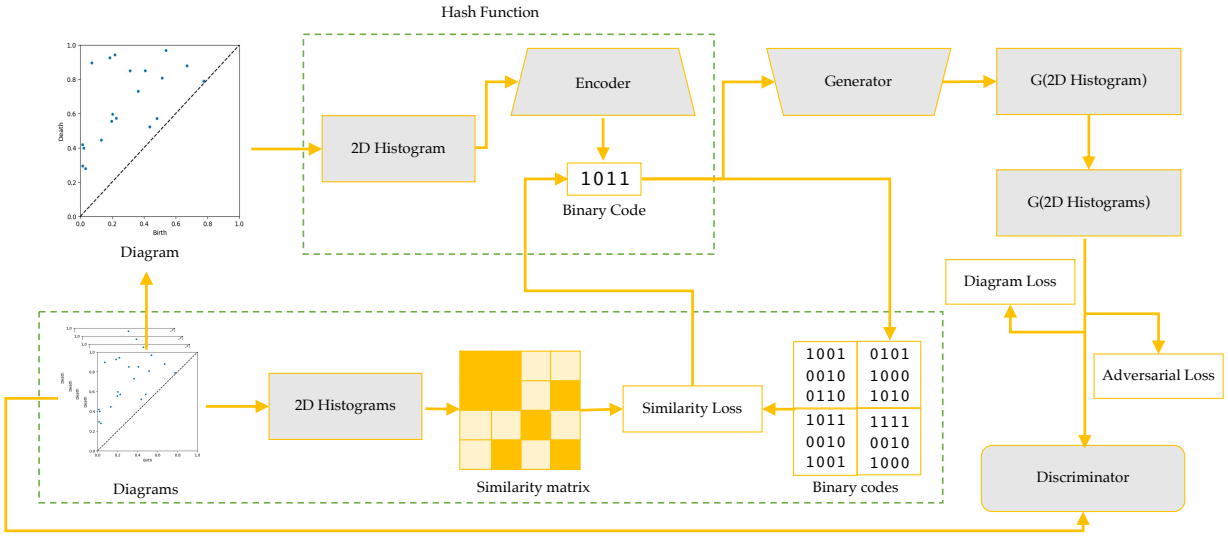

Fig. 2: Architecture of our PD-GAN model to learn filtered topology binary codes. It contains two networks: an encoder network with a hash layer and generative adversarial network (GAN) with a generator and a discriminator.

2.3 Learning to Hash

Given the ability of binary codes to significantly reduce the memory storage requirement and simultaneously boost distance computation for searching, hashing methods have attracted increasing attention for large-scale approximate nearest-neighbor search [3, 41, 42, 56]. In this paper, we focus on using unsupervised machine learning to build a good hashing function that maps high-dimensional data into low-dimensional Hamming space.

Unsupervised building of hash functions can be roughly divided into two groups: non-deep hashing and deep hashing. Typical non-deep hashing includes PCA Hashing (PCAH) [79], Spectral Hashing (SH) [80] and Iterative Quantization (ITQ) [30], which all attempt to preserve a pair-wise similarity of the original data in their resulting binary codes. For example, SH [80] uses eigenfunctions of the data similarity graph to build their hash. More recently, deep hashing [23, 29, 49, 66] has been introduced due to the great advances made in deep learning. The non-linear structure of a convolutional neural network (CNN) could extract multiple hierarchical feature representations of input data and learn their nonlinear relationships to build a binary representation. However, the need for data to be labeled for CNNs meant that unsupervised approaches to learn a hash could not take full advantage of a deep learning model. Inspired by the introduction of the generative adversarial network [31] (GAN), there has been much work on unsupervised learning of hashing functions using a GAN [13, 65] without the need for labeled data. Note that Cao et al.'s [13] usage of Wasserstein distances is not the same as our approach. This previous work uses Wasserstein distances as a loss function inside their GAN, while we will provide a hash that approximates Wasserstein distance for persistence diagrams. Overall, previous hashing approaches mainly focused on the image retrieval tasks, which live in Hilbert space and have nice statistical properties. However, our work on hashing persistence diagrams is in non-Hilbert space. As far as we know, we are the first to transform topological features into a binary representation.

3 Learning Filtered Topology Binary Codes

In this work we will explore the use of concise binary codes to represent persistence diagrams. We take our inspiration from the fact that, due to their stability representing topological structure, many applications only consider larger diagram distances to be significant. This is exactly the space in which hashing operates. Below, we outline our process for building a learned hash function for persistence diagrams. See Fig. 2 for an illustration of the architecture for our approach.

3.1 Vectorized Input

Since our goal is to learn a hash function using a neural network, we first need an intermediate vectorized representation for our persistence diagrams for input into the neural network. We compared popular vectorized approaches outlined in Section 2, including persistence images (PI) and Betti curves (BC), both with L_p-distances, and the 2D histogram (HW) approach to estimating the Wasserstein distance. For all, we use the parameters outlined in their work to create vectorized persistence diagrams. The detailed evaluation process is provided in Section 4.2. As Table 1 illustrates, the HW approach provides the most accurate representation of one-Wasserstein distances compared to other vectorized approaches. Therefore, we adopt its 2D histogram as input for our training and hashing. Similar to their work, we encode our diagrams as histograms on a 2D uniform grid of size 50×50 on $[0,1]^2$. Each cell of the grid counts the number of points in the diagram that lie inside each with an additional cell that contains a count of the total number of off-diagonal points. We additionally augment this representation by reflecting counts across the diagonal to the empty space below as shown in Fig. 3. This reflection is a similar approach as Reininghaus et al. [57]. We have found in the 3D Shape-1 Dataset (Section 4.1) that this augmentation improves the quality of our clustering results by 3% over the standard histogram. In summation, to train our hash from a set of diagrams, $\mathbf{P} = \{p_i\}_{i=1}^N$, we convert them into a set of 2D histograms $\mathbf{V} = \{v_i\}_{i=1}^N$. *For clarity in the exposition, we refer to the general diagrams,* $\mathbf{P}$*, as the input to our approach, but note that these vectorized versions are what are used.*

Table 1: Comparison of vectorized representation using Fowlkes-Mallows score.

Dataset	**N**	$avg(\|P\|)$	HW	PI	BC
3D Shape-1	1,200	63	**0.98**	0.81	0.8

3.2 Similarity Matrix

Next, for training we need an additional input: an $N \times N$ matrix, $\mathbf{S} = \{s_{ij}\}_{i,j=1}^N$, that encodes all pairwise similarities of $\mathbf{P}$. As we outline in the next sections, our learning approach preserves these similarities in Hamming space while building a hash function. In this work we explore two different methods to define similarity. Our first approach is to invert a topological distance measure. The most natural choice

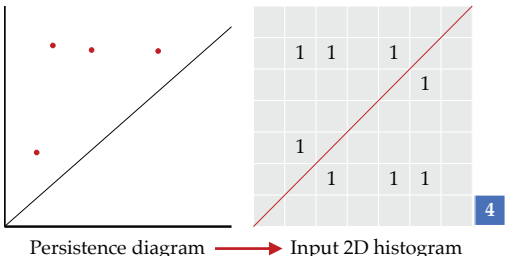

Fig. 3: Our initial vectorized representation used for training and an intermediate step before hashing.

for this direction is to invert 1-Wasserstein distance, since this is the distance that we'd like to mimic in the reduced space. Although, as our experiments in Section 4 illustrate, computing these distances for potentially thousands of training datasets is prohibitively expensive. Therefore, we adopt the Optimal Transport distance used HW [44]. This gives us a computationally feasible way to build the matrix, but also aligns well with our choice in vectorization. Specifically, we compute the matrix such that $s_{ij} = \max(D_{mat}) - d_{ij}$ where D_{mat} is the distance matrix and d_{ij} is the distance between item i and j.

To give more flexibility to the learning algorithm, we also explored the use of a less rigid binary similarity matrix. Rather than using a real-valued measure, with this approach the matrix is formed by simply setting $s_{ij} = 1$ if p_i and p_j are similar and $s_{ij} = -1$ if dissimilar. Defining this similarity threshold can take many forms. We have found a simple threshold on the closest nearest neighbors for each diagram was not sufficient, since this had the tendency to include distant points as similar if a training diagram was not close to many others (also, close diagrams were treated as dissimilar if a diagram had many neighbors). We have opted to use a rejection approach where all diagrams that have distance greater than the mean distance value are rejected as dissimilar in two passes. This is done on a per row (diagram) basis. This approach allows the binary labeling to have a soft threshold that maintains the small distances, discounts large distances, and is less rigid than a nearest neighbors approach. As we show in Section 4, this flexible binary matrix approach outperforms the real-valued distances.

3.3 PD-GAN Model

We now describe how we generate a 64-bit binary code by learning a hash function $h\colon \mathscr{D} \to \{0,1\}^{64}$ where $\mathscr{D}$ is the original space of diagrams. The hash will be designed to maintain topological similarity such that we can $h(\mathbf{P})$ in place of $\mathbf{P}$ for common tasks. As we describe in Section 4, 64 bits was chosen experimentally to be the common code length for all of our testing. To train our hash we use our PD-GAN model that is comprised of two parts:

Encoder and Hashing Layer The Encoder extracts the feature of input diagrams based on a modified VGG19 network [65, 68] with 5 groups of convolution layers with max pooling. The size of filters are 64, 128, 256, 512 and 512 respectively. For details of this configuration we refer to the work of Song et al. [65]. The training of the encoder is driven by minimizing a *similarity loss* function.

The hashing layer generates the binary code. After feeding our input diagrams into the encoder, the binary codes $\mathbf{B} = \{\mathbf{h}(p_i)\}^N$ could be obtained by simply applying a sign function to the last fully connected layer of the encoder. Therefore, this process could be computed as activation function at the end of the encoder network. However, there can be problems with this approach from a vanishing gradient due to the non-smooth sign function. Following the previous work [65], we approximate the sign function with a smoothing method:

$$\mathbf{b} = h(x) = \begin{cases} +1 & \text{for} \quad x \geq 1 \\ x & \text{for} \quad 1 \geq x \geq -1 \\ -1 & \text{for} \quad x \leq -1 \end{cases}$$

GAN: Generator and Discriminator To improve the learning accuracy of the hash function, we use a GAN [31, 65] to evaluate the quality of the generated binary codes. Specifically, the generator $\mathbf{G}$ can be considered as an inverse encoder, where the output of hashing layer is used as the input in a four deconvolutional layered network. $\mathbf{G}$ creates a set of synthetic diagrams, $\mathbf{P}^*$, from only their binary codes. The generator's goal is to create $\mathbf{P}^*$ which can not be discriminated from $\mathbf{P}$ by the discriminator $\mathbf{D}$. $\mathbf{D}$ is trained by minimizing *diagram loss* and *adversarial loss* functions defined in the following subsection. The discriminator informs the generator to improve $\mathbf{P}^*$, while the generator then informs the encoder to improve the hash function.

3.3.1 Loss Functions

We now need to define the loss functions that the above components will try to minimize: similarity loss, diagram loss, and adversarial loss.

Similarity Loss Given the resulting similarity matrix $\mathbf{S} = \{s_{ij}\}_{i,j=1}^N$ from Sec 4.1.1 and the generated binary codes $\mathbf{B} = \{\mathbf{h}(p_i)\}_{i=1}^N$, to build a direct connection between our binary representations and topological distances, we compute the difference between our input similarity matrix and the similarity matrix computed using the produced binary codes. For instance, for real-valued similarities the loss function is defined as:

$$l_{sim} = \frac{1}{2} \sum_{s_{ij} \in \mathbf{S}} (\frac{1}{64} \mathbf{h}(p_i)^T \mathbf{h}(p_j) - s_{ij})$$

Note that given the formulation of binary similarity matrix, this loss function works for either the real valued or binary input case.

Diagram Loss Intuitively, the diagram loss measures on the quality of generated diagrams, $\mathbf{P}^*$, while training compared to the corresponding input diagrams $\mathbf{P}$. Similarly to previous work [65], our diagram loss function is the combination of a pixel-wise Mean Squared Error (MSE), and the perceptual loss computed from the feature map of the lass convolution of the discriminator $\mathbf{D}$, which measure the dis(similarity) between the original $\mathbf{P}$ and generated $\mathbf{P}^*$ based on the ideas of Song et al. [65]. We sum these two losses to form the diagram loss: $l_{dia} = l_{mse} + l_{perceptual}$.

Adversarial Loss The adversarial loss is designed to improve the reconstruction quality of generator $\mathbf{G}$ based on $\mathbf{D}$ and is defined as $l_{adv} = \log(\mathbf{D}(\mathbf{P}) + \log(\mathbf{D}(\mathbf{P}^*)))$.

Combined Loss Finally, the total loss used in training is formulated as the weight sum of the three losses: $l = l_{sim} + \omega_1 l_{dia} + \omega_2 l_{adv}$. The values for ω_1 and ω_2 used in our examples are documented in Section 4.

3.3.2 Learning

To train our PD-GAN, we minimize the loss function in the following steps. We first generate the binary code from the encoder and hashing layer:

$$\mathbf{h}(p_i)^{64} = \mathbf{h}(\mathbf{W}(p_i; \phi))$$

where p_i is an input diagram, ϕ is the vector of encoder parameters, $\mathbf{W}$ is the parameters of the generator $\mathbf{G}$, and $\mathbf{h}$ is the hash function.

After generating $\mathbf{h}(p_i)^{64}$, the generator $\mathbf{G}$ could reconstruct the diagrams $\mathbf{P}^* = \{p^*\}^N$ as: $p_i^* = \mathbf{G}(\mathbf{h}(p_i)^{64}; \theta)$, where θ is the vector of parameters for $\mathbf{G}$. Then the probability of the discriminator $\mathbf{D}$ is denoted as $\mathbf{D}(p_i^*; \sigma)$, where σ is the vector of parameters for $\mathbf{D}$.

In our implementation, we use back-propagation for learning and stochastic gradient descent to find the (locally) optimal parameters for the minimization problem. Specifically, the parameters $\{\phi, \theta, \sigma, \mathbf{W}\}$ be updated during each iteration based on our loss function:

$$\begin{aligned} \mathbf{W} &\leftarrow \mathbf{W} - \tau \bigtriangledown_{\mathbf{W}} (l_{sim} + l_{dia}) \\ \phi &\leftarrow \phi - \tau \bigtriangledown_{\phi} (l_{sim} + l_{dia}) \\ \theta &\leftarrow \theta - \tau \bigtriangledown_{\theta} (l_{dia} + l_{adv}) \\ \sigma &= \sigma + \bigtriangledown_{\sigma} l_{adv} \end{aligned}$$

τ is the learning rate. Details on values used in are examples are documented in Section 4.3.

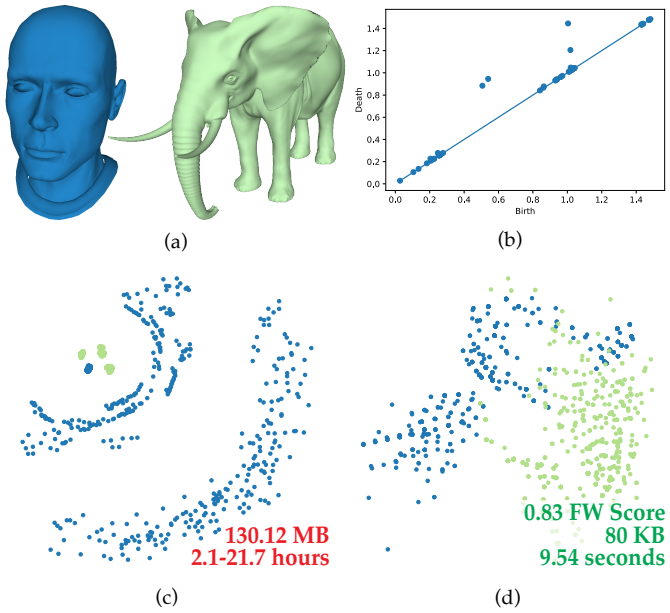

Fig. 4: 3D shape-1: The representative topological clusters are shown (a) by color with an example persistence diagram (b). Two MDS plots show results (c) based on Wasserstein distances and (d) Hamming distance for our generated binary codes. The FW score is the clustering performance measure between two sets of clusters (c) and (d) with range[0,1].

3.4 Binary Encoding and Distance Calculation

After training, diagrams can be hashed by first converting each into their 2D histogram representation and then run through the PD-GAN encoder, mapping each diagram to a 64-bit integer.

A direct consequence of this binary representation is that the representation is concise and that calculating distances in hashed space is a simple Hamming distance computation. This is a simple bit-wise operation: a population count of a XOR of the bits ($popcount(X \oplus Y)$ for binary codes X and Y. Not only is this distance calculation simple, it is also supported in hardware on modern CPUs. In the next section, we show the speed of this calculation in our standard Python implementation, but also with a C++ implementation that leverages this hardware support.

4 Experimental Results

To illustrate the effectiveness of our approach, we use our generated binary representation in clustering applications. We experiments on five datasets and refer to the dataset information in Section 4.1. In order to evaluate the quality of the distance approximations of our approach, we apply a distance-based single-linkage hierarchical clustering algorithm by using the scikit-learn [55] Python library. The objective of single-linkage hierarchical clustering is to produce a nested sequence of partitions by successively merging in a bottom-up fashion. Each observation starts in its own cluster, then pairs of clusters are merged as one moves up the hierarchy until k clusters in total are reached. If a dataset has been shown to cluster in previous work based on topological distance, we adopt the same number of clusters in our experiments. If the number of topologically distinct clusters is not known, we use the elbow method [71] to determine the proper number for k. Results are evaluated against clustering results using the 1-Wasserstein distances. This evaluation is detailed in Section 4.2. Next, we describe how training can be domain-oblivious in Section 4.3. Finally, performance and quality results are reported in Section 4.4.

4.1 Datasets

We used five datasets including 3D shapes, ensemble simulation data, and 2D medical images.

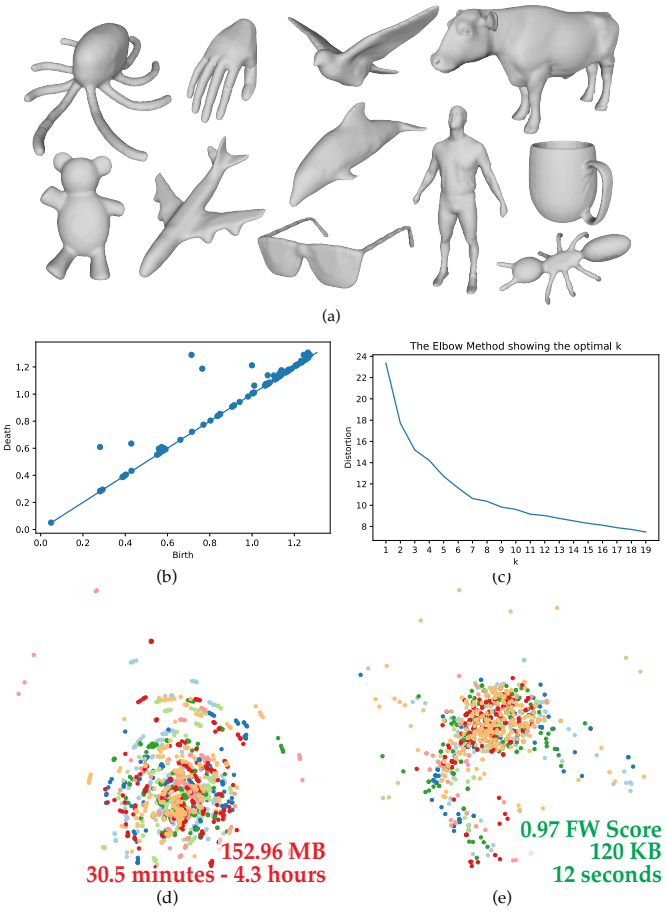

Fig. 5: 3D shape-2 dataset: (a) Example shape meshes from the dataset and (b) one diagram example. As this dataset does not have a known amount of topologically distance clusters, we use (c) the elbow method to find that $k = 7$ clusters exist with respect to the Wasserstein distance. (d) and (e) illustrate the MDS plot for 1-Wasserstein distance and our binary representation distance respectively. A FWS of 0.97 indicates our method almost perfectly match to the original clusters.

3D Shape-1 [67]: this dataset contains 6 different 3D shape classes including camels, horses, elephants, cats, human heads and faces. We create 200 persistence diagrams for each class similarly to Carrière et al. [16] by using a Rips filtration and their implementation. This previous work showed that there were $k = 2$ distinct topological clusters in this dataset. The average number of points in persistence diagrams is 63 with a range from 49 to 100. Fig. 4 shows example shapes and a persistence diagram.

3D Shape-2 [17]: this dataset contains 3D shapes across 19 object categories. 1,900 diagrams are produced in the same way as the previous dataset. The average number of points on this set is 22 with range from 10 to 78. We use elbow method to find the optimal number of clusters is $k = 7$. See Fig. 5.

Vortex Street [76]: this ensemble dataset includes 45 examples of a 2D simulation of flow turbulence behind an obstacle for $k = 5$ clusters of different viscosity. The average number of persistence diagram points in this set is 22 with 20 to 50 each. Diagrams are produced via sublevel set filtration [76] using TTK [72]. Fig. 6 shows examples and a representative persistence diagram.

Starting Vortex [76]: this ensemble dataset includes 12 examples of a 2D simulation with the formation of a vortex behind a wing giving $k = 2$ topological clusters. The average number of points of this set of persistence diagrams is 36 with 30 to 60 each. Diagrams are produced similarly to the previous ensemble. See Fig. 7.

Colorectal Cancer [39]: this is a set of 10,000 regions of interest

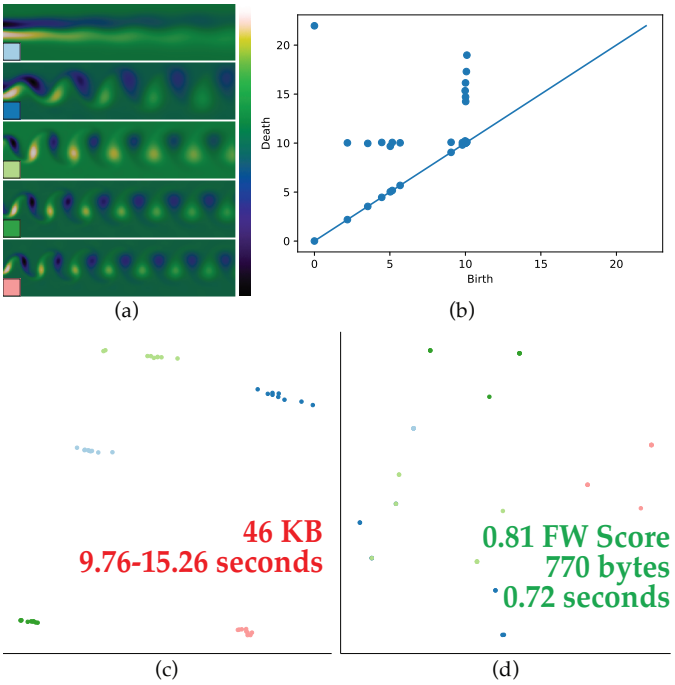

Fig. 6: Vortex Street: This dataset contains $k = 5$ clusters with (a) showing the representative data for each cluster. (b) An example diagram is provided. The MDS plot for this dataset is provided for (c) the 1-Wasserstein distance and (d) Hamming distance of our binary codes.

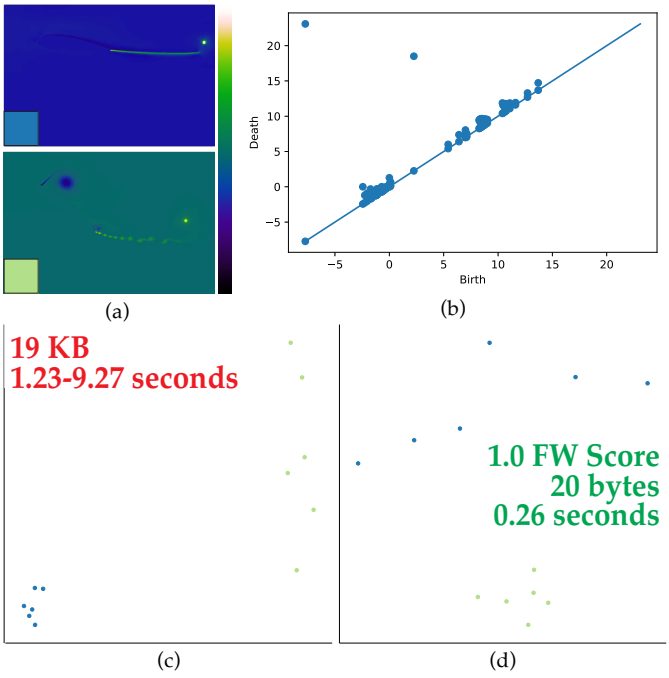

Fig. 7: Starting Vortex: (a) This ensemble set contains 2 clusters. (b) An example persistence diagram is provided. (c and d) A FWS of 1.0 shows that our method achieves a perfect matching among clusters compared clustering using 1-Wasserstein.

images from hematoxylin & eosin (H&E) stained histological images with 9 classes. We deploy the experiment on the full dataset and subset for comparisons, since other approaches cannot run on the full data. The subset contains 200 images per class and produces 1,800 persistence diagrams in total with 503 average number of points ranging from 95 to 789. Diagrams are obtained via sublevel set filtration [45] using Giotto-tda library [69]. We use the elbow method to determine there are $k = 8$ distinct topological clusters. See Fig. 1.

4.2 Evaluation Method

Given a set of input persistence diagrams, $\mathbf{P} = \{p_i\}_{i=1}^N$, our approach computes a set $\mathbf{B} = \{\mathbf{h}(p_i)\}_{i=1}^N$ of binary representations. To evaluate these binary representations, as well as other representations such as persistence images and Betti curves, we compare their clustering performance against clustering based on 1-Wasserstein. As it is considered the standard distance for diagrams, we treat this distance as our ground truth. Let $\mathfrak{C}_1$ be the set of clusters obtained by performing hierarchical clustering on $\mathbf{P}$ using 1-Wasserstein. And let $\mathfrak{C}_2$ be a set of clusters obtained by performing hierarchical clustering on the set of vectorized representations of the persistence diagrams with their associated distances (for example $\mathbf{B}$ uses Hamming distance; Betti curves use Euclidean distance).

We compare these two clusterings $\mathfrak{C}_1$ and $\mathfrak{C}_2$ using the Fowlkes-Mallows score (FMS) [27,51] to quantify the similarity of the clustering. This is defined as the geometric mean of precision and recall:

$$FMS(\mathfrak{C}_1, \mathfrak{C}_2) = \frac{TP}{\sqrt{(TP+FP)(TP+FN)}},$$

where TP (true positive) is the number of pairs of persistence diagrams that belong to the same clusters in $\mathfrak{C}_1$ and $\mathfrak{C}_2$. FP (false positive) is the number of such pairs that are in different clusters in $\mathfrak{C}_1$, but in the same cluster in $\mathfrak{C}_2$. FN (false negative) is the number of such pairs that are in the same cluster in $\mathfrak{C}_1$, but in different clusters in $\mathfrak{C}_2$. An FMS value of 1 means a perfect match with the minimum value being 0. As our experiments show in Section 4.4, our results are all close to 1. In our experiments we use scikit-learn [12] to compute FMS. In addition, multidimensional scaling (MDS) [9] plots are provided in figures to visualize the clustering results of 1-Wasserstein and our approach.

4.3 Domain-Specific vs. Domain-Oblivious Training

Our first approach to training is the obvious one: we train on domain-specific data to evaluate if our hashing function can sufficient preserve this space. We evaluated our approach on the 3D shape-1 dataset, splitting the 1,200 diagrams into training and test sets of size 900 and 300 respectively. Clustering with this trained model gave an FMS of 0.83 when compared to the ground-truth clustering using 1-Wasserstein. While this testing has shown that using domain-data is a viable strategy for our approach, limiting training to domain-specific data would hinder its applicability. As with all machine learning approaches the availability of data is critical to build well-trained models. While datasets like ensemble simulations may have enough data to train the model, this is not guaranteed. Therefore, we evaluated a more general approach.

Ideally, training should be domain-oblivious thereby removing the need for plentiful domain data. To evaluate this possibility, we have trained our model purely on synthetic data. We note that persistence diagrams can be thought of as a specialized 2D scatter plot. Therefore we can produce synthetic diagrams by creating random scatter plots with a uniform point distribution (rejecting points under the diagonal). Training using this naive, synthetic data provided surprising results. We trained our model with diagrams with 50 randomly distributed points and clustered 3D shape-1. The synthetic data produced the same FMS within 0.001 of clustering based on the domain-specific model. In our experiments, we have found that the only requirement for this synthetic approach is that the model should be trained on diagrams with a number of points that is approximately close to the average number of diagram points for the data on which it will be applied. See Table 3 where we show how the number of points in training leads to higher quality results for our 3D shape-1 and 3D shape-2 dataset.

This result is not only not obvious, but one would automatically think the opposite: that a set of naive, randomly diagrams would not be capable sufficiently sampling the space of potential diagrams. Our experimental findings raise interesting questions on this space that we discuss in Section 5.

Table 2: Running times (in seconds) for the approaches outlined in this work to compute the distance matrix of the datasets with **N** diagrams with $avg(|P|)$ points each. 1-Wasserstein (W1), Hera, 2D histograms (HW), progressive Wasserstein (IW), persistence images (PI), Betti Curves (BC), and our approach are provided. The speedup of our approach compared to the next fastest is also provided.

			W1	Hera	HW	IW	PI			BC			Ours			Speedup
Dataset	**N**	$avg(\|P\|)$	Total	Total	Total	Total	Generate	Distance	Total	Generate	Distance	Total	Generate	Distance	Total	
Colon Cancer	10,000	498	–	–	–	–	162.21	1208.8	1371.01	76.36	1257.31	1333.67	135.88	119.08	**254.96**	5.2X
Colon Cancer-sub	1,800	503	>10D	–	>4D	–	31.58	73.04	104.82	4.49	76.59	81.08	9.36	3.98	**13.34**	6.1X
3D Shape-1	1,200	63	78037.24	7523.12	1588.35	–	4.78	38.28	43.06	2.17	35.14	37.31	6.92	2.62	**9.54**	3.9X
3D Shape-2	1,900	22	15321.68	1832.94	633.52	–	6.48	67.4	73.88	3.13	67.69	70.92	8.1	3.9	**12**	5.9X
Vortex Street	45	14	15.26	9.76	0.63	0.09*	0.19	0.041	0.231	0.14	0.04	**0.18**	0.72	0.0033	0.72	-
Starting Vortex	12	36	9.27	1.23	0.14	0.18*	0.06	0.03	0.09	0.02	0.044	**0.064**	0.23	0.026	0.26	-

Table 3: Comparison of clustering results with different training sets using Fowlkes-Mallows score, the Random-20 indicates we use a synthetically random persistence diagram with 20 points each for training, 50 is with 50 points and 100 is with 100 points.

			FMS		
Dataset	**N**	$avg(\|P\|)$	Random-20	Random-50	Random-100
3D Shape-1	1,200	63	0.77	**0.83**	0.67
3D Shape-2	1,900	22	**0.97**	0.96	0.62

We adopt this domain-oblivious approach as the primary method for training in this work. Therefore, we train three models for evaluation: Model-20, Model-50, and Model-100 with 4000 diagrams each with 20, 50, and 100 points per diagram respectively. We set the weighted loss function $\omega_1 = 0.1$ and $\omega_2 = 0.1$ and the default learning rate $\tau = 0.001$. 3D shape-2 and Vortex Street were tested with Model-20. Next, 3D shape-1 and Starting Vortex were tested with Model-50. Finally, both the full Colorectal Cancer dataset and its subset were tested using Model-100. Note the cross-domain applicability of these models.

4.4 Results

All of our experiments were made on Intel Core 3.60GHz × 8 cores (CPU) and Nvidia GeForce GTX 1660 (the GPU was only use for training models) with 32GB of RAM. Our method is implemented in Python with the Tensorflow platform. We also provide a light C++ program for hardware-accelerated Hamming distance computation. All code and data are available in an OSF repository.[1]

We compare the running time and memory usage of our approach with two popular vectorized persistence representations: Persistence Images (PI) [1] and Betti Curves (BC) [28, 59] using GUDHI [70]. In addition, we evaluate our approach against two state-of-the-art approximations to Wasserstein distance: 2D histogram with Optimal Transport (HW) [44] and Progressive Wasserstein (IW) [76] with their implementations. As a ground truth we compare against 1-Wasserstein distance (W1) [19] using scikit-tda [60] and a fast implementation of W1, Hera method [40] using GUDHI [70]. Following the common parameter values for the above. We use a PI bandwidth of $h = 0.02$ with the standard weight function (1/persistence) and the entropic term for HW is $0.1/avg(|P|)$. The grid resolution for all is 50×50 and bounded by the min/max coordinate of diagram. This resolution was determined through experimental evaluation of the range $[10^2, 100^2]$ with a step-size of 1. We found in our testing of 3D shape-1, that sizes over 50 only provided minimal improvement of the FMS (approximately 0.001). Therefore 50 was chosen as the minimum sized representation that still provided good quality results. The PI bandwidth was also determined experimentally by testing the range $[0.001, 1.0]$ with step step size 0.001.

[1] `https://tinyurl.com/ph-to-binarycode`

4.4.1 Speed

Table 2 details our full comparison of runtimes to compute the distance matrix for all pairs in each dataset. This is the input to our chosen clustering approach and is therefore the only point at which each technique differs. We separately list the time to *generate* all representations (e.g., compute persistence images, compute our binary code representation, etc.) and computing the pairwise *distances*. W1, Hera, and IW have no generating time and HW generation time is nominal compared to the costly distance calculation, therefore these times are presented as total runtimes only. Runtimes for IW are provided but marked with an asterisk since direct comparison is not possible. They use a fast C++ implementation (compared to our Python) and compute distances while clustering. Therefore the distance calculation cannot be separated. Parallelism was allowed for vectorized generation, but distances were computed serially. This was chosen to highlight and simplify the runtime comparisons. As the distance computation is embarrassingly parallel, all approaches should be parallelizable with similar comparisons.

Let us first consider the runtimes for W1, Hera, and HW. As this table illustrates, these approaches are prohibitively slow. So much so, it was not feasible to run them on our largest dataset with 10k diagrams. Note, that we were not able to finish the Hera run for the Colorectal Cancer subset before the deadline. As our approach can use hardware acceleration but is not yet in this evaluation, we limit HW to run on the CPU only. Although, we not that in their work [44] they showed that it still takes roughly $40 - -80$ minutes for k-means clustering even with GPU acceleration on 5k persistence diagrams with $50 - -100$ points. Next we evaluated the IW approach. As this is a progressive approach, we ran to completion. As the table illustrates, this approach is extremely fast for the small test datasets. Although in our testing this approach quickly reached the memory limits of our test system (32 GB) and therefore did not scale to our larger datasets (Colorectal Cancer, 3D shape-1 and 3D shape-2). We ran IW on a 50 image subset of Colon Cancer dataset and it took over an hour.

Comparing to other vectorized approaches, our method offers significant performance improvements for our larger datasets giving speedups of $3.9X \sim 6.1X$ when compared to the fastest alternative approach. As the table shows, our distance calculation is incredibly fast leaving the bottleneck of our approach to be the generation of the binary codes. As such, for datasets that are small, where distance calculation does not dominate, our runtimes are comparable to other vectorized approaches.

As a final illustration of the speed, we have implemented the distance calculation of our approach in C++ and leverage the population count and XOR support that exists on modern CPUs. These results are provided for our largest datasets in Table 4. The runtimes are 2-3 orders of magnitude faster than our Python implementation and take all-pairs distance computation down to just milliseconds. Overall, these results show the speed and scalability of our proposed method and how well positioned our binary codes would be for large-scale tasks or databases.

Table 4: Timings for comparisons using a C++ implementation that leverages hardware acceleration for compute Hamming distances.

Dataset	**N**	Python	C++	Speedup
Colon Cancer	10,000	119.08s	161.61ms	737X
3D Shape-1	1,200	2.62s	2.27ms	1154X
3D Shape-2	1,900	3.9s	5.32ms	733X

4.4.2 Memory and Storage

We also compare the memory and storage requirements to save each diagram representation. The 2D points of the diagrams (PD) were saved using 64-bit precision. The grid for PI/BC were also saved with 64-bit precision. We compare this requirement to our approach that saves a single 64-bit number and present this results in Table 5 for all of our datasets. Reduction rate is compared to the next smallest representation. As this table illustrates, as one would expect, saving a single integer has significant benefits for storage and memory. The encoder size for our approach, which would also have to be saved as well, is not included in this results. As our approach is domain-oblivious, a single model can be potentially used on a multitude of datasets from a large number of domains. Therefore, the cost of storing the encoder would be amortized in practice. Our approach can not only reduce storage overheads, but also produces high-quality topological similarity measures as shown the next subsection.

Table 5: Comparison of size (in MB) with persistence summaries. As vectorized summaries of persistence diagrams, here PI and BC have same vector size (50x50).

Dataset	**N**	$avg(\|P\|)$	PD	PI/BC	Ours (64bits)	Reduction
Colon Cancer	10,000	498	800.96	201.21	**0.64**	192X
Colon Cancer-sub	1,800	503	144.96	36.2	**0.12**	254X
3D Shape-1	1,200	63	130.12	24.13	**0.08**	301X
3D Shape-2	1,900	22	152.96	36.2	**0.12**	301X
Vortex Street	45	14	0.046	0.9	**0.00077**	1168X
Starting Vortex	12	36	0.019	0.24	**0.00002**	1200X

4.4.3 Quality

Table 6 shows the evaluation of cluster quality between our results and other methods, using the evaluation method described in Section 4.2. The quality is determined by the FMS between the clusterings obtained for the different persistence diagram representations compared to the clustering produced when using 1-Wasserstein distances directly on the input persistence diagrams. As such, we could only run comparisons on an 1,800 diagram subset of the colorectal cancer dataset since running the full dataset was not possible (W1, Hera) or failed (IW) for some approaches. For our approach, we provide the FMS for our two methods for forming a similarity matrix: real-valued or binary similarity. As this table illustrates, our approach provides comparable or better quality results when compared to progressive Wasserstein (IW), persistence images (PI), or Betti Curves (BC). We time-limit IW to the total runtime of our approach as reported in Table 2. IW would, in time, converge to the exact 1-Wasserstein distance. Therefore for a fair comparison, we only look at their quality for the same amount of running time. Not only are the results from the binary codes on par with other approaches, our approach almost achieves perfect reproduction of the clustering for the Colorectal Cancer and 3D Shape-2 datasets. Moreover, it perfectly matches the clustering of Starting Vortex. Note these results use our domain-oblivious training approach and therefore the same models were applied to more than one type of data.

Next, we evaluate quality in FMS in relation to the number of bits used in the binary code. We experimented with the clustering result of 3D shape-1 by varying bit lengths from 24-128. The results of this experiment are provided in Table 7. As this table shows, and as one would expect, increases in bit length result in increases in quality of the final result. In the end, we found 64-bit provides high-quality results and is a length that provides simpler implementations than higher bit counts.

Table 6: Comparison of clustering results using the Fowlkes-Mallows score, as described in Section 4.2. Scores range from 0 to 1; a score of 1 indicates identical clusters. The clusterings using different persistence diagram representations (and their distances) are compared to the 1-Wasserstein-based clustering of the input persistence diagrams.

					Ours		
Dataset	$avg(\|P\|)$	IW	PI	BC	Model	Real	Bin
Colon-sub	503	0.71	0.98	**0.99**	100	0.92	**0.99**
3D Shape-1	63	0.54	0.8	0.81	50	0.82	**0.83**
Starting Vort.	36	**1**	**1**	**1**	50	0.63	**1**
3D Shape-2	22	0.74	**0.97**	0.96	20	0.91	**0.97**
Vortex Street	14	**1**	0.79	0.78	20	0.80	0.81

Table 7: Comparison of clustering results with different number of bits, using the Fowlkes-Mallows score.

Dataset	**N**	$avg(\|P\|)$	24 bits	48 bits	64 bits	128 bits
3D Shape-1	1,200	63	0.64	0.75	0.83	0.84

Table 8 illustrates an FMS comparison of clustering the 3D Shape-1 dataset with different similarity matrix strategies. In this table, DMat is the real-value distance matrix. S-X denotes the use of a binary similarity matrix built from the real-value distance matrix. Specifically, S-1 shows the strategy using a fixed number of k-nearest neighbors where $k = 1000$ out of the 4000 training set. S-2 has $k = 600$ and S-3 has $k = 1400$. For a soft threshold similarity matrix, S-4 uses a strategy of limiting similarity to use a global threshold of 25% percent. Finally, S-5 is our two pass mean rejection. As this figure illustrates, the two pass approach leads to more accurate clustering and is therefore used by our work.

Table 8: Comparison of clustering results with different similarity matrix computation methods, using the Fowlkes-Mallows score.

Trained Model	**N**	$avg(\|P\|)$	DMat	S-1	S-2	S-3	S-4	S-5
Model-50	4,000	50	0.82	0.77	0.78	0.8	0.81	**0.83**

5 Conclusions

In the paper, we present an approach to produce concise binary codes of persistence diagrams that maintain topological similarity. The key to this approach is the training of a machine learning model that learns a hash, not on domain-specific data, but on randomly generated 2D scatter plots. This leads to a technique that is domain-oblivious, where a model can be applied across multiple domains or types of data without the need for retraining. As this is a hashing approach, our technique is not likely to maintain small distances. For applications where close distances are discounted, our approach is well-suited. It is still an open question if a hashing approach could be designed such that small distances are maintained. The data used in our synthetically trained model only needs to roughly match the average number of diagram points of the testing dataset. In practice, we have found this is not an overly strict requirement. For instance, the Colorectral Cancer dataset has on average 500 points, but Model-100 worked well in our tests. In regards to storage, while our binary code is small, one would still need to save the encoder, which for deep networks can be many MB. As we mentioned previously, given that a single model can be applied to many datasets across many domains, this amortized cost would be negligible in practice. Finally, this work illustrated the benefits of this

representation through examples from topological clustering, where our new binary codes provide fast, high-quality results. Moreover, the potential scalability of such an approach was highlighted through the low storage requirements of the binary codes along with extremely fast distance calculations using on-chip acceleration.

ACKNOWLEDGMENTS

Removed for anonymity.

REFERENCES

[1] H. Adams, T. Emerson, M. Kirby, R. Neville, C. Peterson, P. Shipman, S. Chepushtanova, E. Hanson, F. Motta, and L. Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18, 2017.

[2] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 2000.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer science (FOCS'06)*, pp. 459–468. IEEE, 2006. https://doi.org/10.1145/1327452.1327494.

[4] R. Anirudh, V. Venkataraman, K. Natesan Ramamurthy, and P. Turaga. A Riemannian framework for statistical analysis of topological persistence diagrams. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 68–76, 2016. https://arxiv.org/abs/1605.08912.

[5] E. Berry, Y.-C. Chen, J. Cisewski-Kehe, and B. T. Fasy. Functional summaries of persistence diagrams. *Applied and Computational Topology*, 4:211–262, 2020. Also available at arXiv:1804.01618.

[6] D. P. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, 1981. https://doi.org/10.1016/S0305-0548(97)00019-1.

[7] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer. Topoms: Comprehensive topological exploration for molecular and condensed-matter systems. *Journal of Computational Chemistry*, 39(16):936–952, 2018. https://doi.org/10.1002/jcc.25181.

[8] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392(1-3):5–22, 2008. https://doi.org/10.1016/j.tcs.2007.10.018.

[9] I. Borg and P. J. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer Science & Business Media, 2005.

[10] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2010. https://doi.org/10.1109/TVCG.2010.253.

[11] P. Bubenik. Statistical topological data analysis using persistence landscapes. *J. Mach. Learn. Res.*, 16(1):77–102, 2015.

[12] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[13] Y. Cao, B. Liu, M. Long, and J. Wang. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1287–1296, 2018.

[14] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003. https://doi.org/10.1016/S0925-7721(02)00093-7.

[15] M. Carriere, M. Cuturi, and S. Oudot. Sliced Wasserstein kernel for persistence diagrams. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 664–673. JMLR. org, 2017. https://dl.acm.org/doi/10.5555/3305381.3305450.

[16] M. Carrière, S. Y. Oudot, and M. Ovsjanikov. Stable topological signatures for points on 3d shapes. In *Computer Graphics Forum*, vol. 34, pp. 1–12. Wiley Online Library, 2015.

[17] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009.

[18] Y.-C. Chen, D. Wang, A. Rinaldo, and L. Wasserman. Statistical analysis of persistence intensity functions, 2015. arXiv preprint arXiv:1510.02502.

[19] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence-diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007. https://doi.org/10.1145/1064092.1064133.

[20] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, and Y. Mileyko. Lipschitz functions have L_p-stable persistence. *Foundations of Computational Mathematics*, 10(2):127–139, 2010.

[21] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in Neural Information Processing Systems*, 26:2292–2300, 2013.

[22] L. De Floriani, U. Fugacci, F. Iuricich, and P. Magillo. Morse complexes for shape segmentation and homological analysis: Discrete models and algorithms. *Computer Graphics Forum*, 34(2):761–785, 2015. https://doi.org/10.1111/cgf.12596.

[23] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *European Conference on Computer Vision*, pp. 219–234. Springer, 2016.

[24] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010. https://doi.org/10.1007/978-3-540-33259-6_7.

[25] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 454–463. IEEE, 2000. https://doi.org/10.1007/s00454-002-2885-2.

[26] B. T. Fasy, X. He, Z. Liu, S. Micka, D. L. Millman, and B. Zhu. Approximate nearest neighbors in the space of persistence diagrams. *arXiv preprint arXiv:1812.11257*, 2018. https://arxiv.org/abs/1812.11257.

[27] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

[28] M. Gameiro, K. Mischaikow, and W. Kalies. Topological characterization of spatial-temporal chaos. *Physical Review E*, 70(3):035203, 2004.

[29] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen. Video captioning with attention-based lstm and semantic consistency. *IEEE Transactions on Multimedia*, 19(9):2045–2055, 2017.

[30] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2012. https://doi.org/110.1109/TPAMI.2012.193.

[31] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Stat*, 1050:10, 2014.

[32] D. Günther, R. A. Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny. Characterizing molecular interactions in chemical systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2476–2485, 2014. https://doi.org/10.1109/TVCG.2014.2346403.

[33] A. Gyulassy, P.-T. Bremer, R. Grout, H. Kolla, J. Chen, and V. Pascucci. Stability of dissipation elements: A case study in combustion. *Computer Graphics Forum*, 33(3):51–60, 2014. https://doi.org/10.1111/cgf.12361.

[34] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci. A practical approach to morse-smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008. https://doi.org/10.1109/TVCG.2008.110.

[35] J.-C. Hausmann. On the Vietoris-Rips complexes and a cohomology theory for metric spaces. In *Prospects in Topology: Proceedings of a Conference in Honor of William Browder*, vol. 138 of *Annals of Mathematics Studies*, pp. 175–188. Princeton University Press, 1995. https://doi.org/10.1515/9781400882588-013.

[36] G. Henselman and R. Ghrist. Matroid filtrations and computational persistent homology, 2016. arXiv preprint arXiv:1606.00199.

[37] L. V. Kantorovich. On the translocation of masses. *Journal of Mathematical Sciences*, 133(4):1381–1382, 2006.

[38] J. Kasten, J. Reininghaus, I. Hotz, and H.-C. Hege. Two-dimensional time-dependent vortex regions based on the acceleration magnitude. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2080–2087, 2011. https://doi.org/10.1109/tvcg.2011.249.

[39] J. N. Kather, N. Halama, and A. Marx. 100,000 histological images of human colorectal cancer and healthy tissue, Apr. 2018. doi: 10.5281/zenodo.1214456

[40] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare

persistence diagrams. *Journal of Experimental Algorithmics (JEA)*, 22:1–20, 2017. https://doi.org/10.1145/3064175.

[41] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems*, pp. 1042–1050, 2009. https://dl.acm.org/doi/10.5555/2984093.2984211.

[42] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th International Conference on Computer Vision*, pp. 2130–2137. IEEE, 2009. https://doi.org/10.1109/ICCV.2009.5459466.

[43] G. Kusano, K. Fukumizu, and Y. Hiraoka. Kernel method for persistence diagrams via kernel embedding and weight factor. *The Journal of Machine Learning Research*, 18(1):6947–6987, 2017.

[44] T. Lacombe, M. Cuturi, and S. Oudot. Large scale computation of means and clusters for persistence diagrams using optimal transport. In *Advances in Neural Information Processing Systems*, pp. 9770–9780, 2018. https://arxiv.org/abs/1805.08331.

[45] P. Lawson, A. B. Sholl, J. Q. Brown, B. T. Fasy, and C. Wenk. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Scientific Reports*, 9(1):1–15, 2019. https://doi.org/10.1038/s41598-018-36798-y.

[46] T. Le and M. Yamada. Persistence Fisher kernel: A Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, pp. 10028–10039, 2018. arXiv preprint arXiv:1802.03569.

[47] H. Lee, M. K. Chung, H. Kang, B.-N. Kim, and D. S. Lee. Discriminative persistent homology of brain networks. In *2011 IEEE international symposium on biomedical imaging: from nano to macro*, pp. 841–844. IEEE, 2011. https://doi.org/10.1109/ISBI.2011.5872535.

[48] H. Lee, M. K. Chung, H. Kang, and D. S. Lee. Hole detection in metabolic connectivity of Alzheimer's disease using k-Laplacian. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 297–304. Springer, 2014. https://doi.org/10.1007/978-3-319-10443-0_38.

[49] K. Lin, J. Lu, C.-S. Chen, and J. Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1183–1192, 2016.

[50] D. Maljovec, B. Wang, P. Rosen, A. Alfonsi, G. Pastore, C. Rabiti, and V. Pascucci. Topology-inspired partition-based sensitivity analysis and visualization of nuclear simulations. *Proc. of IEEE PacificVis*, 2016. 10.1109/PACIFICVIS.2016.7465252.

[51] M. Meila. Comparing clusterings – an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[52] Z. Meng, D. V. Anand, Y. Lu, J. Wu, and K. Xia. Weighted persistent homology for biomolecular data analysis. *Scientific Reports*, 10(1):1–15, 2020. https://doi.org/10.1002/cnm.2914.

[53] G. Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[54] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust online computation of reeb graphs: simplicity and speed. *ACM Transactions on Graphics (TOG)*, 26(3):58–es, 2007. https://dl.acm.org/doi/10.1145/1276377.1276449.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[56] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*, pp. 1509–1517, 2009. https://dl.acm.org/doi/10.5555/2984093.2984263.

[57] J. Reininghaus, S. Huber, U. Bauer, and R. Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4741–4748, 2015. https://doi.org/10.1109/CVPR.2015.7299106.

[58] B. Rieck, F. Sadlo, and H. Leitte. Topological machine learning with persistence indicator functions. pp. 87–101, 2017.

[59] V. Robins. Computational topology for point data: Betti numbers of α-shapes. In *Morphology of Condensed Matter*, pp. 261–274. Springer, 2002.

[60] N. Saul and C. Tralie. Scikit-tda: Topological data analysis for python, 2019. doi: 10.5281/zenodo.2533369

[61] D. R. Sheehy and S. Sheth. Sketching persistence diagrams. In *Symposium on Computational Geometry*, 2021. arXiv preprint arXiv:2012.01967.

[62] D. Shnier, M. A. Voineagu, and I. Voineagu. Persistent homology analysis of brain transcriptome data in autism. *Journal of the Royal Society Interface*, 16(158):20190531, 2019. https://doi.org/10.1098/rsif.2019.0531.

[63] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Lifted Wasserstein matcher for fast and robust topology tracking. In *2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 23–33. IEEE, 2018.

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

[65] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen. Binary generative adversarial networks for image retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[66] J. Song, T. He, L. Gao, X. Xu, and H. T. Shen. Deep region hashing for efficient large-scale instance search from images. *arXiv preprint arXiv:1701.07901*, 2017.

[67] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on graphics (TOG)*, 23(3):399–405, 2004.

[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[69] G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. Medina-Mardones, A. Dassatti, and K. Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2020.

[70] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 3.4.1 ed., 2021.

[71] R. L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[72] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The topology toolkit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2017.

[73] K. Turner, Y. Mileyko, S. Mukherjee, and J. Harer. Fréchet means for distributions of persistence diagrams. *Discrete & Computational Geometry*, 52(1):44–70, 2014.

[74] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[75] P. M. Vaidya. Geometry helps in matching. *SIAM Journal on Computing*, 18(6):1201–1225, 1989.

[76] J. Vidal, J. Budin, and J. Tierny. Progressive Wasserstein barycenters of persistence diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):151–161, 2019. https://doi.org/10.1109/TVCG.2019.2934256.

[77] L. Vietoris. Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen. *Mathematische Annalen*, 97(1):454–472, 1927. https://doi.org/10.1007/BF01447877.

[78] H. Wagner, C. Chen, and E. Vuçini. Efficient computation of persistent homology for cubical data. In *Topological Methods in Data Analysis and Visualization II*, pp. 91–106. Springer, 2012.

[79] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012.

[80] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, pp. 1753–1760, 2009. https://dl.acm.org/doi/10.5555/2981780.2981999.

[81] K. Xia and G.-W. Wei. Persistent homology analysis of protein structure, flexibility, and folding. *International journal for numerical methods in biomedical engineering*, 30(8):814–844, 2014. https://doi.org/10.1002/cnm.2655.

[82] X. Xu, J. Cisewski-Kehe, S. B. Green, and D. Nagai. Finding cosmic voids and filament loops using topological data analysis. *Astronomy and Computing*, 27:34–52, 2019.

[83] Q. Zhao and Y. Wang. Learning metrics for persistence-based summaries and applications for graph classification. *arXiv preprint arXiv:1904.12189*, 2019.