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

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

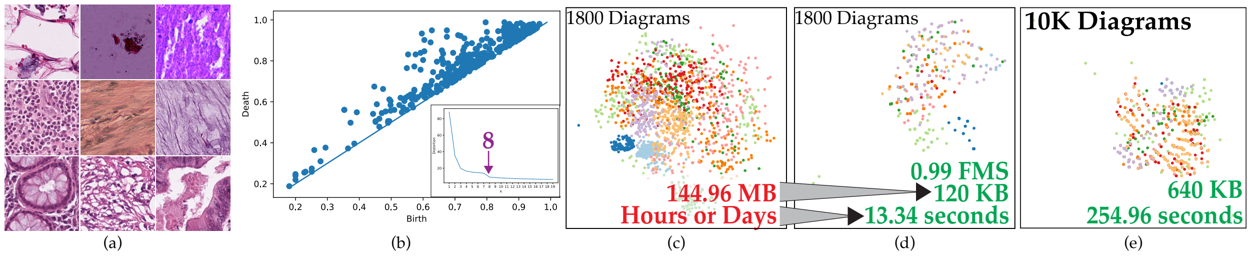


Fig. 1: (a) Examples from a dataset of 10k histology images of colorectal cancer. (b) An example persistence diagram that encodes the topological structure in an image. The inset illustrates an elbow method plot run from clustering a subset of 1800 images using 1-Wasserstein distance. This shows there are approximately 8 topologically distinct clusters. (c) Clustering result using 1-Wasserstein distance on the subset. (d) Our high-quality concise representation uses only a fraction of the memory and computation time. (e) Our approach scales to 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 using standard 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 for 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 the potential of 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) [23] have been shown to express the fundamental structure of scalar fields in a way that is generally applicable to many domains. TDA approaches—such as persistent homology [24], contour trees [12], Reeb graphs [8, 55], and Morse(–Smale) complexes [21, 33]—have demonstrated ability to extract meaningful structure in a variety of research applications, including 3D shape matching [14], combustion physics [10, 32], nuclear physics [49], fluid dynamics [38], chemistry [7, 31], Alzheimer's disease [47], autism spectrum disorders [46, 63], cancer histology [44], protein folding [81], and bio-molecular analysis [52]. More generally, recent work includes using TDA quantification as input to machine learning [13, 44, 58, 59].

As described in detail in Section 2.1, a persistence diagram is a common way to present the topological structure in a dataset. The distance between these diagrams is often used to measure the topological (dis)similarity between data, which has important applications in scientific visualization [83]. Moreover, for approaches that cluster based on topological similarly [14, 44, 74, 76], computing the distance between diagrams is a fundamental operation. However, computing these distances is costly in practice.

The most widely accepted persistence diagram distance measures, the Wasserstein distances, require expensive matching of all persistence points between two diagrams. As discussed in Section 2, to combat this complexity, many approaches have attempted 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 for expressing topological structure that is more concise than previous works, but also leads directly to faster computations of distances. In particular, we show how to reduce diagrams to simple 64-bit binary codes. 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 persistence points as are in the test data. In other words, the 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, distances are calculated by a simple bit-wise count comparison between binary codes (the Hamming distance). This makes distance computation extremely fast and scalable. We illustrate this scaling through the clustering of a

• *Yu Qin is with Tulane University. E-mail: yqin2@tulane.edu.*
• *Brittany Terese Fasy is with Montana State University. 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

dataset with 10k diagrams, a size which is not achievable for several existing approaches.

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, potentially 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, we view it as a topological space or a sequence of (nested) topological spaces, called a *filtration*. Then, we employ homology and persistent homology, respectively, to qualitatively and quantitatively describe it. Homology is a concept from algebraic topology that captures the fundamental structure of a topological space [54]. The structure is qualitatively described through the homology groups, whose generators we call *features*. Each feature has a dimension associated to it; dimension zero features are connected components, dimension one features are loops or tunnels, dimension two features in $\mathbb{R}^3$ correspond to voids, etc. Persistent homology quantifies the homology of an entire filtration [23]. In particular, each feature f_i also has a birth time b_i and a death time d_i, indicating the parameters of the filtration for which that feature "lives"; we call the difference $d_i - b_i$ the *lifetime* or *persistence* of the feature. We represent this feature as the persistence point $(b_i, d_i) \in \mathbb{R}^2$. Since a feature must be born before it dies with respect to the filtration parameter, persistence points are restricted to lying above the *diagonal* defined by the line $x = y$.[1] The *persistence diagram* is the multi-set of birth-death pairs.

This abstract concept is best illustrated by describing example filtrations. For scalar fields, a common filtration is the evolution of sublevel sets. The sublevel set filtration is the progression of a watershed transformation [6], where water sources grow from local minima (i.e., basins) of the field. The zero-dimensional features (i.e., connected components) are the watersheds that begin at the local minima. One-dimensional features form when a watershed completely surrounds a local maximum (i.e., peak). The lifetimes of these features are recorded as the scalar value (i.e., water height) from where a feature first appears (*birth*) to the value at which it merges with an "older" feature (*death*). All birth and death events occur at critical points.

For unstructured points (i.e., point cloud data that is given as a discrete set of points with a pairwise distance defined), a filtration can be built from the evolution of a Vietoris–Rips complex [34, 77]. Here, simplices are formed in the complex from an ever-increasing neighborhood around each point. In this filtration, keeping track of the connected components can detect the size and the number of distinct clusters and recording the evolution of one-dimensional features can detect the presence and size of circular features in the data. These features are agnostic to the domain of the data or even the dimension of the space. For this reason, one can say that these features represent the fundamental structure of data.

Fig. 2 illustrates a simple example of the zero-dimensional features of a sublevel set filtration on a scalar field with two basins. As the sublevel set increases, first the purple feature is born in the deepest basin, followed by the green feature. As the filtration continues, the

[1]In general, it is possible for some features to have a birth but no associated death. In our experiments, these features are not as informative as the features with defined birth and death times. For this reason, we do not include them in our definition above.

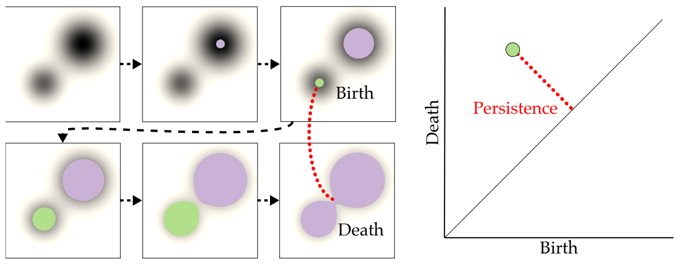


Fig. 2: Progression of a sub-level set of a scalar field. The feature (green) is born at a minimum and dies when it merges with an older feature (purple). The birth and death are represented as a point in the zero-dimensional persistence diagram. The L_1 distance from this point to the diagonal is the lifetime, or persistence, of the feature (red).

green feature eventually dies when it merges with the purple connected component. The green feature is represented in the diagram by plotting these birth and death values. Note, that since the purple feature has not yet died, it is not yet in the diagram.

Wasserstein and Bottleneck Distances Letting $\mathscr{D}$ denote the collection of all diagrams, the q-Wasserstein distance $d_q : \mathscr{D} \times \mathscr{D} \to \mathbb{R}$ is defined by

$$d_q(p_1, p_2) := \min_M \left(\sum_{(a,b)\in M} ||a-b||_\infty^q + \frac{1}{2^{q-1}} \sum_{a \in M^c} |a_x - a_y|^q \right)^{1/q},$$

where M ranges over all matchings between persistence diagrams p_1 and p_2, and M^c is the set of persistence points in $p_1 \sqcup p_2$ that do not appear in the matching M; see [19, 37, 40, 53]. The Wasserstein distance optimizes a matching between two diagrams and sums the distances between matched points (M) as well as the point-to-diagonal distances for unmatched points (M^c). Letting $q \to \infty$, gives the bottleneck distance (or, interleaving distance) [18]. Setting $q = 1$ gives the one-Wasserstein distance (W1), a popular choice in applications and therefore the target of our work [1, 13]. Note that when we refer to the Wasserstein distance without specifying q, we are referring to $q = 1$. These diagrams are (Lipschitz) stable in the presence of slight perturbations or noise in data [19]. Given this stability, diagrams that are close in distance are often considered to be topologically similar. Computationally, however, both the Wasserstein and the bottleneck distances are expensive, as they require computing the optimal matching between persistence points in the two diagrams. In particular, computing the Wasserstein distance between two diagrams takes $O(n^2 \log^2 n)$ time, where n is the total number of simplices (which, in turn, can be exponential in the size of the input data) [75].

Approximating Distances When many distances between diagrams need to be computed, the roughly quadratic computation can be daunting. Thus, several approaches have been introduced to approximate computing the Wasserstein distances [2, 5, 40, 62, 64]. One approach that is quite successful is to simplify the input representation, before a persistence diagram is even computed [35, 78]. However, this makes assumptions on the underlying domain (e.g., a 2D or 3D image).

Kerber et al. [40] introduced an approximate Wasserstein distance algorithm to accelerate the computation of the matching using a k-d tree. This iterative computation bounds either quality or time to approximate the Wasserstein distances; we call this distance the progressive Wasserstein (PW) distance. This algorithm was extended by Vidal et al. [76] for the problem of computing barycenters of persistence diagrams. Although very fast, these approaches still require the pairwise matching and, as we illustrate in Section 4.4, the memory requirements can be significant as data sizes grow.

In many circumstances, the diagrams we have are bounded, that is, there exists a square $D \subset \mathbb{R}^2$ such that all persistence points lie inside D. Then, for a given input parameter $d \in \mathbb{N}$, we create a $d \times d$ grid over D. Using this grid, we define a histogram, where we count the number of persistence points in each grid cell. Fasy et al. [25] use these histograms to design a data structure that supports searching for near neighbors

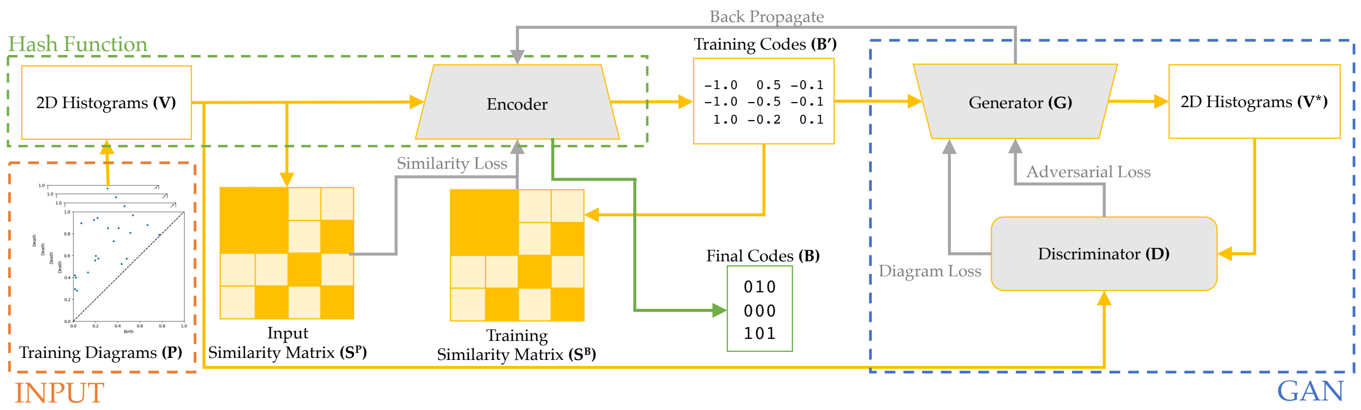


Fig. 3: Training architecture of the PD-GAN model that takes a set of persistence diagrams **P** as input and learns corresponding binary codes **B**. Yellow arrows depict flow of diagrams through training, while grey arrows represent information flow (loss functions, etc.). A Generative adversarial network (GAN) is used to train the encoder of binary codes such that the similarity matrix of codes, $\mathbf{S}^B$, closely matches the similarity of the input diagrams, $\mathbf{S}^P$. Our final hash function that takes in a diagram and produces a binary code is highlighted in green.

based on the bottleneck distance and Lacombe et al. [43] computes Wasserstein distances (optimal transport in this space) between the histograms. While this work can benefit from several fast optimal transport computation approaches [20, 65], it still poses significant costs for distance computations. We call the Wasserstein distance between histograms the Histogram Wasserstein (HW) distance.

2.2 Other Topological Descriptors

The previous approaches for histograms can be thought of as representations of the distribution of the persistence points. In this line of research, density estimators built over the persistence points have been studied by several groups of researchers [1, 4, 16, 58, 59]. One benefit of considering density estimators is that they can be vectorized. 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.

Persistence 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]). Roughly, these images estimate the density of points by summing a Gaussian kernel centered at each point. 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, 16, 84].

Betti Curves (BC) Other topological invariants include the Euler characteristic and the Betti numbers. For a given integer $k \in \mathbb{N}$, the k^{th} Betti curve (BC) [27, 60, 82] is the rank of the dimension k homology group with respect to the filtration parameter. Roughly, this is the count of the topological features present as the filtration parameter increases. Each feature associated to a persistence point (b_i, d_i) contributes $+1$ to the Betti curve in the interval from b_i to d_i. Hence, the more persistent a feature, the more it contributes to the Betti curve. The L_p distance between Betti curves can be computed explicitly, or can be approximated by sampling or discretizing the domain. Often, the latter approach is preferred in practice. Hence, using the Betti curves requires selecting the dimension(s) of interest, a bounding box, and a discretization resolution.

2.3 Learning to Hash

Given the ability of binary codes to significantly boost distance computation for searching, hashing methods have attracted increasing attention for large-scale approximate nearest-neighbor search [3, 41, 42, 57]. In this paper, we focus on using unsupervised machine learning to build a good hash 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 [79], spectral hashing [80], and iterative quantization [29], which all attempt to preserve a pairwise similarity of the original data in their resulting binary codes. For example, spectral hashing uses eigenfunctions of the data similarity graph to build their hash. More recently, deep hashing [22, 28, 48, 67] has been introduced due to the great advances made in deep learning. The non-linear structure of a convolutional neural network (CNN) can 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 means that unsupervised approaches to learn a hash function cannot take full advantage of a deep learning model. Inspired by the introduction of the generative adversarial network (GAN) [30], other work focuses on the unsupervised learning of hash functions using a GAN without the need for labeled data [11, 66]. Overall, previous hashing approaches mainly focused on the image retrieval tasks, which live in Hilbert space and have nice statistical properties. In our work, we show that a natural image hashing approach [66] can be used to hash persistence diagrams in non-Hilbert space. In doing so, we present the first approach to transform topological features into a binary representation.

3 Learning Topological Binary Codes

In this work, we explore the use of concise binary codes to represent persistence diagrams. Below, our process takes as input N persistence diagrams $\mathbf{P} = \{p_i\}_{i=1}^N$ and trains neural networks to hash persistence diagrams to binary codes. Then, the set of binary codes and their Hamming distances can be used in lieu of persistence diagrams and their Wasserstein distances. See Fig. 3 for an illustration of the architecture for our approach.

3.1 Vectorizing Input

The first step in our hash function is to take as input a set of diagrams $\mathbf{P} = \{p_i\}_{i=1}^N$ and to convert it into a vectorized form appropriate to use as input to train networks. As mentioned in Section 2, we have several choices for vectorized representations of the input persistence diagrams, including 2D histograms using Wasserstein distance (HW) [43], persistence images using L_2 distance (PI) [1], and Betti curves using L_2 distance (BC) [27]. Using the parameters outlined in the respective papers, we compare the use of these three vectorizations for clustering relative to clustering persistence diagrams using Wasserstein distance. Table 1 provides the results using the Fowlkes-Mallows score (FMS), the evaluation measure used in this work (see Section 4.2). The scores can be in the interval $[0, 1]$, with a score of 1 indicating a perfect match. As Table 1 illustrates, HW provides the most accurate clustering. Therefore, we use 2D histogram vectorization for our training and hashing.

We transform our diagrams to histograms on a 2D uniform grid of size 50×50 on $[0, 1]^2$ with the entropic term: $0.1/avg_{i=1}^N|p_i|$ (the recommended parameter values [43]). Each cell of the grid counts the number of persistence points in the diagram that lie inside each, with an additional cell that contains a count of the total number of persistence points. In addition, similar to Reininghaus et al. [58], we augment this representation by reflecting counts across the diagonal to the empty space below, see Fig. 4. We found that this augmentation improves the quality of our clustering results over the standard histogram (improvement of 3% for the 3D Shape-1 dataset of Section 4.1). In summary, the first step of training our hash function is to convert each diagram $p_i \in \mathbf{P} = \{p_i\}_{i=1}^N$ into a 2D histogram v_i. In Fig. 3, $\mathbf{V}$ denotes the set of all such histograms.

Table 1: Comparison of vectorized representations using Fowlkes-Mallows score. Values closer to one are better.

Dataset	HW	PI	BC
3D Shape-1	**0.98**	0.81	0.8

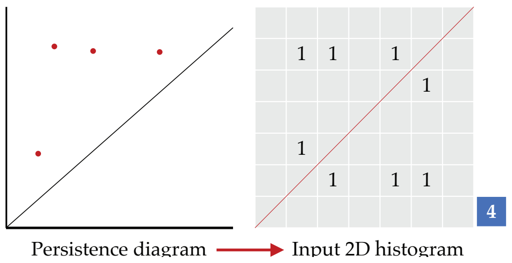


Fig. 4: The initial 2D histogram representation used for training and an intermediate step before hashing.

3.2 Similarity Matrix

Before training our model, we need one more object: a $N \times N$ similarity matrix, $\mathbf{S}^P = \{s_{ij}^P\}_{i,j=1}^N$, that stores all pairwise similarities between histograms in $\mathbf{V}$. We explore two different methods to define similarity. Our first approach is to invert a topological distance measure. The most natural choice is to invert Wasserstein distance, since this is the distance that we would like to mimic using Hamming distances of binary codes. As our experiments in Section 4 illustrate, computing these distances for potentially thousands of training datasets is prohibitively expensive. Therefore we adopt the HW distance on $\mathbf{V}$ as used in Lacombe et al. [43]. This gives us a computationally feasible way to build the matrix, and also aligns well with our choice of intermediate vectorization. Specifically, we compute a real-valued similarity matrix with $s_{ij}^P := 1 - d_1(v_i, v_j)$.

To give more flexibility to the learning algorithm, we also explored the use of a less rigid binary similarity matrix. Rather than using the real/original values above, the matrix is formed by setting $s_{ij}^P = 1$ if v_i and v_j are similar and $s_{ij}^P = -1$ if they are dissimilar. Defining this similarity can take many forms. We found that thresholding based on the closest k-nearest neighbors for each diagram was not sufficient. This had the tendency to count distant persistence diagrams 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 opted to use a rejection approach where all diagrams that have distance greater than the mean distance value are rejected as dissimilar. This is done in two passes on a per row (diagram) basis. This approach allows the binary labeling to have a soft threshold that maintains small distances, discounts large distances, and is less rigid than a nearest neighbor approach. As we show in Section 4, this flexible binary matrix approach outperforms 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 is designed to maintain topological similarity when comparing binary codes with Hamming distance. Our learning framework uses the image hashing approach of Song et al. [66]. Below we describe their approach and how it is used in our context. To build our hash function, two key parts are trained for the PD-GAN model, the encoder and the GAN; see Fig. 3:

Encoder The encoder extracts the features of input diagrams based on a pretrained VGG19 network [66,69] with five groups of convolution layers with max pooling. The number of filters in each of these groups are 64, 128, 256, and 512. The output size of the last fully connected layer is the bit length $L = 64$. The training of the encoder is driven by minimizing a *similarity loss* function; see Section 3.3.1. After training, each binary code, $\mathbf{h}(v_i) = b_i \in \mathbf{B}$, can be formed from the signs of the values of the last fully connected layer when v_i is run through the encoder (i.e., $\mathbf{h}(v_i)^k = sgn(x^k)$, where x^k is the k-th value in the last layer). We call this last layer the binary-code layer. However, a non-smooth, binary representation can be problematic for the gradient computation needed in training. To avoid this problem, an intermediary, real representation of the binary code, $\mathbf{B}' = \{\mathbf{h}'(v_i)\}_{i=1}^N$, is used during training:

$$\mathbf{h}'(v_i)^k = \begin{cases} +1 & \text{for} \quad x^k \geq 1 \\ x^k & \text{for} \quad 1 \geq x^k \geq -1 \\ -1 & \text{for} \quad x^k \leq -1, \end{cases}$$

where k is the k-th element of the binary code. After training, the binary codes, $\mathbf{B}$, can be extracted from the binary-code layer as described above.

GAN: Generator and Discriminator To improve the accuracy of the learned hash function, a GAN [30, 66] is used. See Fig. 3 (blue). Specifically, the generator $\mathbf{G}$ can be considered as an inverse encoder, where the output of the encoder is used as the input to the network with four deconvolutional layers. $\mathbf{G}$ creates a set of synthetic histograms, $\mathbf{V}^*$, from their training codes, $\mathbf{B}'$. The generator's goal is to create a $\mathbf{V}^*$ which cannot be distinguished from $\mathbf{V}$ by the discriminator $\mathbf{D}$. $\mathbf{G}$ is trained by minimizing *diagram loss* and *adversarial loss* functions defined below. The discriminator informs the generator to improve $\mathbf{V}^*$, while the generator then informs the encoder to improve the hash function.

3.3.1 Loss Functions

Loss functions need to be defined for the above components to minimize: similarity loss, diagram loss, and adversarial loss.

Similarity Loss Given the similarity matrix $\mathbf{S}^P = \{s_{ij}^P\}_{i,j=1}^N$ from Section 3.2 and the training codes $\mathbf{B}'$, the similarity loss captures a direct connection between our binary representations and topological distances. Let $\mathbf{S}^B = \{s_{ij}^B\}_{i,j=1}^N$, where $s_{ij}^B = \frac{1}{L}\mathbf{h}'(v_i)^T\mathbf{h}'(v_j)$ and L is the bit length. Then the similarity loss is defined as:

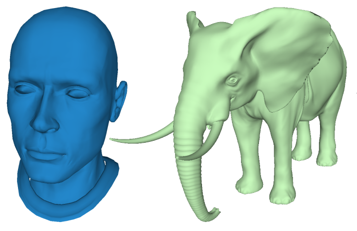
$$l_{sim} = \frac{1}{2}||\mathbf{S}^B - \mathbf{S}^P||^2 + ||\mathbf{B}' - \mathbf{B}||^2 \, ,$$

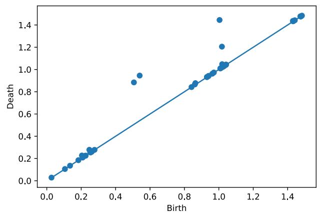
where $||.||$ are Euclidean norms.

Diagram Loss Intuitively, the diagram loss compares the generated histograms, $\mathbf{V}^*$, to the corresponding input histograms, $\mathbf{V}$. The diagram loss function is the combination of a pixel-wise Mean Squared Error (MSE), and the perceptual loss [66]. Perceptual loss is given by the last layer of the discriminator, $\mathbf{D}$. Perceptual loss accounts for the observation [45] that pixel-wise MSE optimization often lacks high-frequency content. We verified experimentally in our test dataset that including perceptual loss provided more accurate results. These two losses are summed to form the diagram loss: $l_{dia} = l_{mse} + l_{perceptual}$.

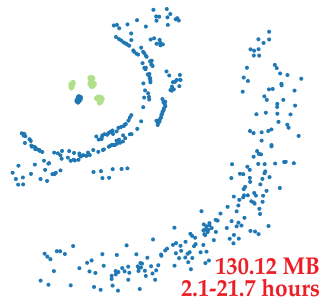
Adversarial Loss The adversarial loss is designed to improve the reconstruction quality of generator $\mathbf{G}$ and is defined as $l_{adv} = \log(\mathbf{D}(\mathbf{V})) + \log(1 - \mathbf{D}(\mathbf{V}^*))$.

Combined Loss Finally, the combined loss used in training is the weighted sum of the three losses: $l = l_{sim} + \omega_1 l_{dia} + \omega_2 l_{adv}$, where $\omega_1 = \omega_2 = 0.1$ were used in our experiments as in Song et al. [66].

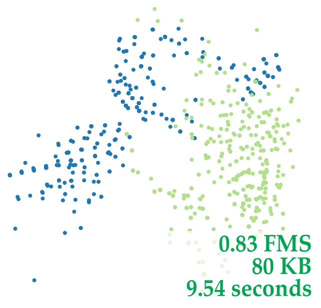

(a)



(b)



(c)



(d)

Fig. 5: 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 distance and (d) Hamming distance for our generated binary codes. The FMS score is the clustering performance measure between two sets of clusters (c) and (d).

3.3.2 Learning

To train the PD-GAN, the loss functions are minimized in the following steps. First, the training codes $\mathbf{B}'$ are created by the encoder, using parameters ϕ for the VGG19 part of the encoder and $\mathbf{W}$ for the binary-code layer. Then the generator $\mathbf{G}$ reconstructs the diagrams $\mathbf{V}^*$ with a parameter vector θ. The discriminator $\mathbf{D}$ uses the parameter vector σ. Back-propagation for learning and stochastic gradient descent are used to find the (locally) optimal parameters based on the loss functions. Specifically, the parameters $\{\phi, \theta, \sigma, \mathbf{W}\}$ are updated during each iteration, where $\tau = 0.001$ is the default learning rate in our experiments:

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

3.4 Hash Function and Distance Computation

After training, diagrams can be hashed by first converting them into their 2D histogram representation and then running each through the PD-GAN encoder to map each to a 64-bit integer. See Fig. 3 (green). A direct consequence of this binary encoding is that the representation is concise and distances between codes are computed in Hamming space. Distance computations are now a simple bit-wise operation: a population count of the XOR of the bits ($popcount(X \oplus Y)$ for binary codes X and Y). Not only is this distance computation simple, it is also supported in hardware on modern CPUs. In Section 4 we show the speed of this computation 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 experimented 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 [56] Python library. It takes a *distance matrix* as input, which contains all pairwise distances between histograms. The objective of single-linkage hierarchical clustering is to produce a nested sequence of partitions by successively merging clusters in a bottom-up fashion until k clusters in total are reached.

For the datasets that have undefined k topological clusters, we use the elbow method [72] to determine the number of clusters. For this we deploy the hierarchical clustering for a sequential set of potential values for k and then plot the total within-cluster sum of square distances versus k (which measures the compactness of the clustering). The final number k of clusters is then chosen by the elbow of the curve.

Results are evaluated against clustering results using the Wasserstein distance. Test datasets are described in Section 4.1. Evaluation methods are 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 evaluated the clustering of five datasets that include 3D shapes, ensemble simulation data, and 2D medical images.

3D Shape-1 [68]: This dataset contains 6 different 3D shape classes including camels, horses, elephants, cats, human heads, and faces. We created 200 persistence diagrams for each class using the implementation of Carrière et al. [14] to produce a Vietoris–Rips filtration. This previous work showed that there were $k = 2$ distinct topological clusters in this dataset. The average number of persistence points per diagram is 63, ranging from 49 to 100. Fig. 5 shows example shapes and a persistence diagram.

3D Shape-2 [15]: 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 persistence points per diagram on this set is 22, ranging from 10 to 78. We use the elbow method [72] to find that $k = 7$ clusters exist. See Fig. 6.

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 points in this set is 22, ranging from 20 to 50. Diagrams are produced via sublevel set filtration [76] using TTK [73]. Fig. 7 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 persistence points of this set of persistence diagrams is 36 with 30 to 60 each. Diagrams are produced similarly to the previous ensemble. See Fig. 8.

Colorectal Cancer [39]: This is a set of 10,000 regions of interest images from hematoxylin & eosin (H&E) stained histological images with 9 classes. The average number of persistence points in this set is 498, ranging from 78 to 802. We conduct experiments on the full dataset and on a subset, 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 an average of 503 persistence points ranging from 95 to 789. Diagrams are obtained via sublevel set filtration [44] using the Giotto-tda library [70]. We use the elbow method [72] to determine that there are $k = 8$ distinct topological clusters. See Fig. 1.

4.2 Evaluation of Clustering

Given a set of input persistence diagrams, $\mathbf{P}$, our approach computes a set $\mathbf{B}$, 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 using Wasserstein distance. 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 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).

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 average persistence points (Avg Pts). Wasserstein (W1), Hera, 2D histograms (HW), progressive Wasserstein (PW), 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	PW	PI			BC			Ours			Speedup
Dataset	**N**	Avg Pts	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	>7D	>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 persistence points each for training, 50 is with 50 persistence points and 100 is with 100 persistence points.

			FMS		
Dataset	**N**	Avg Pts	Model-20	Model-50	Model-100
3D Shape-2	1,900	22	**0.97**	0.96	0.62
3D Shape-1	1,200	63	0.77	**0.83**	0.67
Colon Cancer-sub	1,800	503	0.71	0.76	**0.99**

We compare these two clusterings $\mathfrak{C}_1$ and $\mathfrak{C}_2$ using the Fowlkes-Mallows score (FMS) [26,51] 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 $\mathfrak{C}_2$. A 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 addition, multidimensional scaling (MDS) [9] plots are provided in Fig. 5-8 to visualize clustering results using 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 hash function can sufficiently 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 a FMS of 0.83 when compared to the ground-truth clustering using Wasserstein. While this test shows 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 persistence 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 persistence points and clustered 3D shape-1. The synthetic data produced the same FMS within 0.001 of clustering using the domain-specific model. In our experiments, we found that the only requirement for this approach is that the synthetic diagrams used in training should each have a number of points close to the average number of persistence points in the data to be clustered. This result is not only not obvious, but one would automatically think the opposite: that a set of naive, random diagrams would not sufficiently sample the space of potential diagrams. Our experimental findings raise interesting questions on this space that we discuss in Section 5.

We adopt this domain-oblivious approach as the primary method for training in this work. We train three models for evaluation: Model-20, Model-50, and Model-100 with 4000 diagrams each with 20, 50, and 100 persistence points per diagram respectively. Table 3 illustrates the requirement described above where matching the number of persistence points in training diagrams to the average number of points in the clustered data leads to higher quality results. In our experiments, 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 used for training models) with 32GB of RAM. Our method is implemented in Python with the Tensorflow platform using the implementation[2] of Song et al. [66] for our training architecture. We also provide a lightweight C++ program for hardware-accelerated Hamming distance computation. Our code and data are available in an OSF repository.[3]

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) [27, 60] using GUDHI [71]. In addition, we evaluate our approach against two state-of-the-art approximations of Wasserstein distance: 2D histograms with Optimal Transport (HW) [43] and Progressive Wasserstein (PW) [76] with their implementations. As a ground truth we compare against Wasserstein distance (W1) [18] using scikit-tda [61] and a fast implementation of W1, the Hera method [40] using GUDHI [71], 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 $0.1/avg_{i=1}^{N}|p_i|$. The grid resolution for all is 50×50 and bounded by the min/max coordinate of the 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 provides good quality results. The PI bandwidth was also determined experimentally by testing the range $[0.001, 1.0]$ with step size 0.001.

[2]`https://github.com/ht014/BGAN`
[3]`https://osf.io/q58c3/`

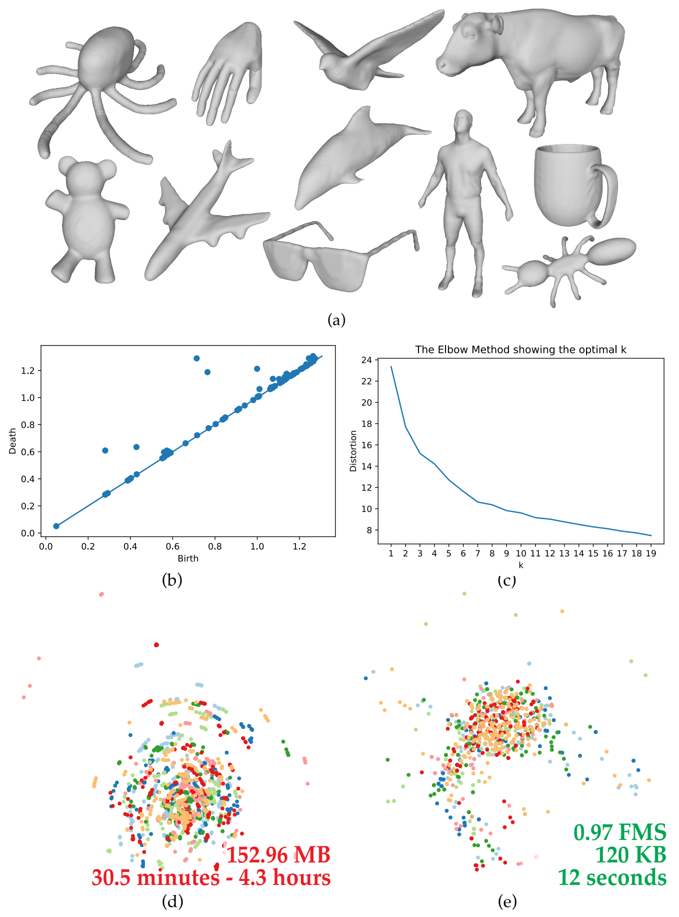


Fig. 6: 3D shape-2 dataset: (a) Example shape meshes from the dataset and (b) one diagram example. As this dataset does not have a known amount of topologically distinct clusters, we use (c) the elbow method to find that $k = 7$ clusters exist with respect to the Wasserstein distance. (d) and (e) illustrate the MDS plots for Wasserstein distance and for Hamming distance for our generated binary codes, respectively. The FMS of 0.97 indicates that our method almost perfectly matches the original clusters.

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 single-linkage clustering 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 to compute the pairwise *distances*. W1, Hera, and PW have no generation time, and the generation time for HW is nominal compared to the costly distance computation; therefore these times are presented as total runtimes only. Runtimes for PW 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 generation of the vectorizations, 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. In fact, it was not feasible to run them on our largest dataset with 10k diagrams. For a fair comparison to our Python distance calculation, we do not run HW using GPU acceleration. We note that Latombe et al. [43] showed

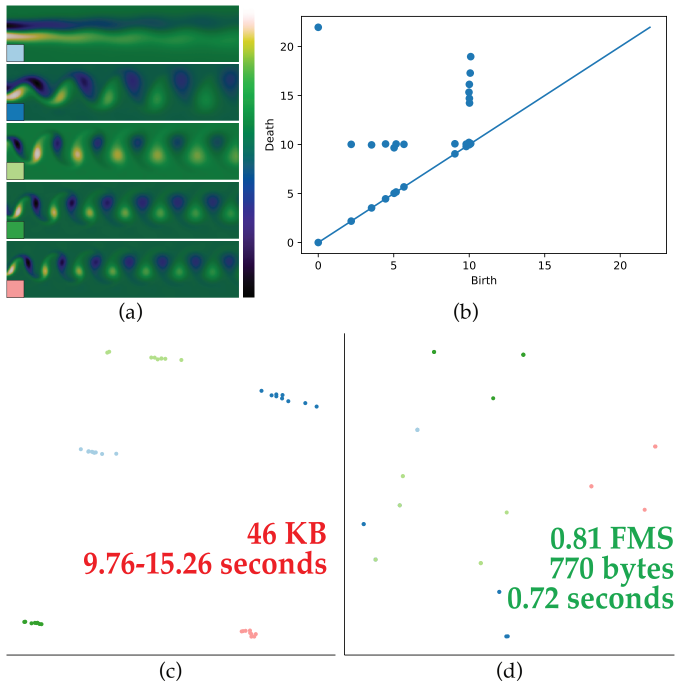


Fig. 7: Vortex Street: This dataset contains $k = 5$ clusters with (a) showing the representative data for each cluster. (b) An example diagram is provided. The MDS plot for this dataset is provided for (c) the Wasserstein distance and (d) the Hamming distance of our binary codes.

that HW still takes roughly $40 - 80$ minutes for k-means clustering even with GPU acceleration on 5k persistence diagrams with $50 - 100$ persistence points. Next we evaluated the PW approach. This is a progressive approach, and for a fair comparison we ran it to completion. As the table illustrates, this approach is extremely fast for the small test datasets. Although in our testing the author's implementation of this approach quickly reached the memory limits of our system (32 GB) and therefore did not scale to our larger datasets (Colorectal Cancer, 3D shape-1 and 3D shape-2). We ran PW on a 50 image subset of the 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 computation 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 computation 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 to 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

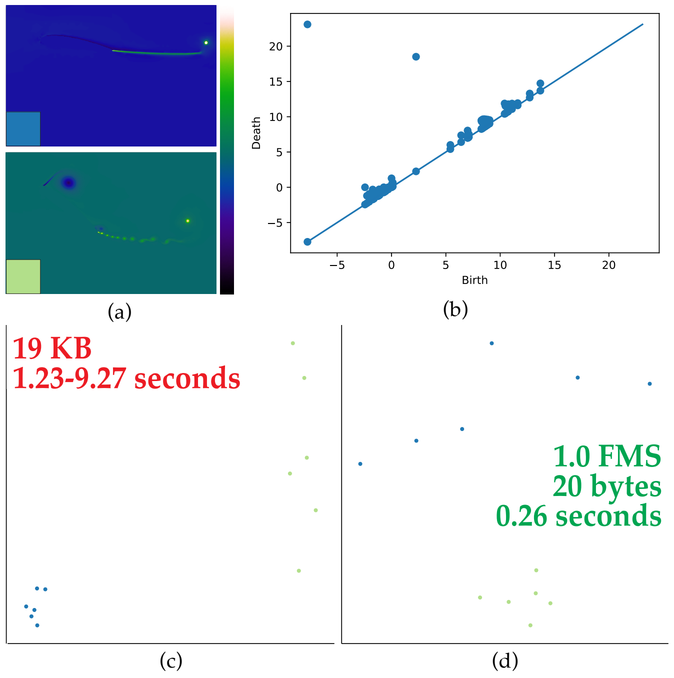


Fig. 8: Starting Vortex: (a) This ensemble set contains 2 clusters. (b) An example persistence diagram is provided. (c) and (d) The FMS of 1.0 shows that our method achieves a perfect matching among clusters compared to clustering using Wasserstein.

4.4.2 Memory and Storage

Table 5 describes the potential memory and storage gains from our representation. The 2D persistence points of the diagrams (PD) were saved using 64-bit precision. The grids for PI/BC were also saved with 64-bit precision. We compare this requirement to our approach that saves a single 64-bit number. Reduction rate is compared to the next smallest representation. As this table illustrates, as one would expect, saving a single integer can have significant benefits for storage and memory. The encoder size for our approach, which would also have to be saved, is not included in these results. This cost can vary. Our encoder for this work was the same as Song et al. [66]: a standard, pre-trained VGG19 [69] model, which is approximately 500MB. However, it may be possible to compress this size [17] . For instance, work has shown VGG19 can reduce its parameters $5-20X$ [50]. Other models like SqueezeNet [36] reduce to less than 0.5MB. Overall, since our approach is domain-oblivious, a single model can be potentially used on a multitude of datasets from a large number of domains. Thus the cost of storing the encoder would be amortized in practice and will not grow as data sizes increase. Therefore this approach has the potential to greatly reduce storage overheads as the use of TDA grows.

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

Dataset	N	Avg Pts	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 clustering quality, comparing our clustering results with 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 Wasserstein distance directly on the persistence diagrams. As such, we could only run comparisons on a 1,800 diagram subset of the colorectal cancer dataset since running the full dataset was not possible (W1, Hera) or failed (PW) 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 (PW), persistence images (PI), or Betti Curves (BC). We time-limit PW to the total runtime of our approach as reported in Table 2. PW would, in time, converge to the exact 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.

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 Wasserstein-based clustering of the input persistence diagrams.

					Ours		
Dataset	Avg Pts	PW	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**
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
Starting Vortex	36	**1**	**1**	**1**	50	0.63	**1**

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

Dataset	N	Avg Pts	24 bits	48 bits	64 bits	128 bits	256 bits
3D Shape-1	1,200	63	0.64	0.75	**0.83**	0.84	0.86
3D Shape-2	1,900	22	0.82	0.89	**0.97**	0.97	0.98
Vortex Street	45	14	0.67	0.77	**0.81**	0.83	0.86
Starting Vortex	12	36	0.94	1	**1**	1	1

To further evaluate the quality of distance preservation using our binary codes, we provide scatterplots in Fig. 9 by setting Wasserstein and Hamming distance as x-y coordinates for each point pair. To avoid overdrawing, each point is drawn as a Gaussian kernel density estimator. Each point in the plot is a diagram with the horizontal position given by the W1 distance and vertical being the Hamming distance. If distances are being maintained, this plot should be linear about a diagonal. As this figure illustrates, this is the case for our binary codes. In Fig. 9 (a) the points are provided as well (yellow). This shows there is clear separation in both dimensions (dotted line) meaning that these clusters are well-maintained in both distances. Finally, Fig. 9 (b) illustrates the plot for Vortex Street, an example with a lower FMS. As highlighted with red arrows, there are clear clusters of points where distance is not being maintained and likely give the lower score (although still higher quality than PI and BC).

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, 3D shape-2, Vortex Street, and Starting Vortex by varying

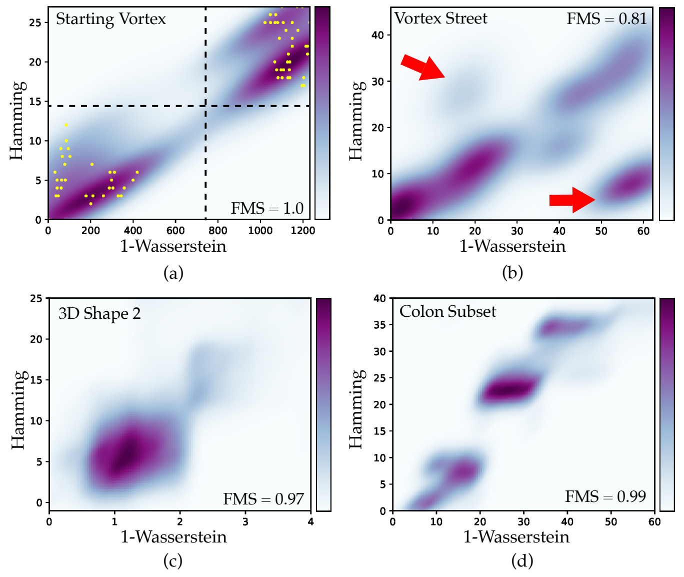


Fig. 9: (a-d) Plots to illustrate distance preservation using binary codes for various datasets. Each is a scatterplot of points for each diagram whose horizontal position is W1 distance and vertical is Hamming. Each is drawn with a kernel density estimator. All exhibit a linear, diagonal shape meaning that distances are being preserved. (a) also plots the points directly (yellow). There is a clear separation of the two clusters both horizontally and vertically. (b) highlights the two areas that break this linear property that likely give its lower FMS.

bit lengths from 24-256. 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, although with a noticeable falloff in gains after 64 bits. Therefore we opted for 64-bit codes since they provide high-quality results with simpler implementations than the higher bit counts.

Table 8 illustrates a FMS comparison of clustering the 3D Shape-1 dataset with different similarity matrix strategies. In this table, Real denotes the real-valued similarity matrix. S-X denotes the use of a binary similarity matrix built from the real-valued distance matrix. S-1 uses fixed number of k-nearest neighbors, where $k = 1000$ out of the 4000 training set. Similarly, S-2 uses $k = 600$ and S-3 uses $k = 1400$. 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 Pts	Real	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 persistence points of the testing dataset. In practice, we have found this is not an overly strict requirement. For instance, the Colorectral Cancer dataset has 500 persistence points on average, 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, given that a single model can be applied to many datasets across many domains, we argue that the amortized cost could be nominal in practice. Although, we plan to explore how to reduce this overhead in future work. 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 scalability of such an approach was highlighted through the potential low storage requirements of the binary codes along with extremely fast distance computations using on-chip acceleration.

Acknowledgments

This work was supported by the National Science Foundation and the National Institutes of Health (NSF DMS 1664848 and 1664858), NSF CCF 2046730, and NSF IIS 2136744.

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.

[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.

[5] D. P. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, 1981.

[6] S. Beucher and C. Lantuejoul. International workshop on image processing: Real-time edge and motion detection/estimation, September 1979. lecture notes.

[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.

[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.

[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.

[11] 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.

[12] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[13] 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.

[14] 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.

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

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

[17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[18] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistenced-diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[19] 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.

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

[21] 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.

[22] 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.

[23] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[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.

[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.

[41] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems*, pp. 1042–1050, 2009.

[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.

[43] 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.

[44] 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.

[45] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. M. Alvarez. Domain-adaptive deep network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4289–4297, 2017.

[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.

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

[54] J. R. Munkres. *Algebraic Topology*. Prentice Hall, Upper Saddle River, NJ, 1964.

[55] 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.

[56] 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.

[57] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*, pp. 1509–1517, 2009.

[58] 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.

[59] B. Rieck, F. Sadlo, and H. Leitte. Topological machine learning with persistence indicator functions. In *Topological Methods in Data Analysis and Visualization*, pp. 87–101. Springer, 2017.

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

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

[62] D. R. Sheehy and S. Sheth. Sketching persistence diagrams. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss

Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] J. Song, T. He, L. Gao, X. Xu, and H. T. Shen. Deep region hashing for generic instance search from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

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

[69] 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.

[70] G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. M. Medina-Mardones, A. Dassatti, and K. Hess. giotto-tda:: A topological data analysis toolkit for machine learning and data exploration. *J. Mach. Learn. Res.*, 22:39–1, 2021.

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

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

[73] 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.

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

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

[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.

[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.

[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] L. Yan, T. B. Masood, R. Sridharamurthy, F. Rasheed, V. Natarajan, I. Hotz, and B. Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Computer Graphics Forum*, 40(3):599–633, 2021. doi: 10.1111/cgf.14331

[84] Q. Zhao and Y. Wang. Learning metrics for persistence-based summaries and applications for graph classification. *Advances in Neural Information Processing Systems*, 32:9859–9870, 2019.