Mapper Interactive: A Scalable, Extendable, and Interactive Toolbox for the Visual Exploration of High-Dimensional Data

Youjia Zhou* Nithin Chalapathi† Archit Rathore‡ Yaodong Zhao§ Bei Wang¶

Scientific Computing and Imaging (SCI) Institute, University of Utah

ABSTRACT

The mapper algorithm is a popular tool from topological data analysis for extracting topological summaries of high-dimensional datasets. In this paper, we present *Mapper Interactive*, a web-based framework for the interactive analysis and visualization of high-dimensional point cloud data. It implements the mapper algorithm in an interactive, scalable, and easily extendable way, thus supporting practical data analysis. In particular, its command-line API can compute mapper graphs for 1 million points of 256 dimensions in about 3 minutes (4 times faster than the vanilla implementation). Its visual interface allows on-the-fly computation and manipulation of the mapper graph based on user-specified parameters and supports the addition of new analysis modules with a few lines of code. *Mapper Interactive* makes the mapper algorithm accessible to nonspecialists and accelerates topological analytics workflows.

1 INTRODUCTION

The mapper algorithm, first introduced by Singh *et al.*, is a popular tool from topological data analysis (TDA) for extracting topological summaries of high-dimensional datasets in the form of simplicial complexes [19]. It is rooted in the idea of "partial clustering of the data guided by a set of functions defined on the data" [19]. In many practical scenarios, the 1D skeletons of such simplicial complexes – the mapper graphs – serve as simple descriptions of the data and capture important information about their topological structures.

From a theoretical perspective, researchers are actively studying the mapper algorithm and its properties (e.g., [2, 4, 15]). From an implementational perspective, a few open-sourced tools exist that implement the mapper algorithm and support data analysis, including *KeplerMapper* [24], *giotta-tda* [21], *Gudhi* [23], and *Python Mapper* [14]. *Mapper Interactive* focuses on simultaneously addressing important aspects of the mapper algorithm involving scalability, extensibility, and interactivity in an integrated way, which differentiates it from existing implementations.

Mapper Interactive makes the mapper algorithm accessible to nonspecialists, and those with a passing knowledge of programming concepts and TDA. At the same time, it gives specialists the ability to extend the system by adding analysis and visualization components in a modular fashion.

In summary, we introduce *Mapper Interactive*, a toolbox for the visual exploration of high-dimensional data. It comes with both a command line API for offline computation and a web-based interface for online computation of mapper graphs. Our contributions include:

- **Extendability:** We demonstrate the extendability of our toolbox via simple examples for both novice and expert users.
- **Interactivity:** We provide three case studies that demonstrate the strengths of *Mapper Interactive* in supporting fast insight generation for well-known and new datasets.
- **Scalability:** We present a simple but effective strategy for speeding up mapper graph computations. Such a strategy is applicable to any mapper algorithm implementation using DBSCAN as a subroutine. The command line API of *Mapper Interactive* computes mapper graphs for 1 million points of 256 dimensions in 3 minutes; and it is in general 3 to 6 times faster than the vanilla implementation. The GPU implementation provides an additional 2 times acceleration for 1 million points in comparison to its CPU counterpart.

Mapper Interactive is open source under the MIT license and is available via Github[1].

*e-mail: zhou325@sci.utah.edu

†e-mail: nithin.chalapathi@utah.edu

‡e-mail: archit.rathore@utah.edu

§e-mail: yaodong.fs@gmail.com

¶e-mail: beiwang@sci.utah.edu

2 RELATED WORK

A few open-source implementations of the mapper algorithm are described in the literature. Müllner and Babu implemented *Python Mapper* [14], which computes a mapper graph with a set of predetermined parameters. It contains a graphical user interface (GUI) that interfaces with the library and visualizes the resulting mapper graph. However, it does not provide any interactive analytic features.

Recently, Veen and Saul presented *KeplerMapper* [24], a versatile and user-friendly implementation of the mapper algorithm. *KeplerMapper* provides some limited interactive capabilities in the visual encoding of a single mapper graph. For instance, users can color the nodes of a mapper graph and glean some information regarding the distribution of data points within each node. *KeplerMapper* also includes an adapter for *NetworkX* [8], where users can manually create a visualization of a mapper graph (generated by *KeplerMapper*) using *NetworkX*. Similar to *Python Mapper*, *KeplerMapper* precomputes each mapper graph with a set of predetermined parameters; the resulting visualization is exported as a separate HTML file to be loaded in a browser. However, its mapper implementation does not scale with a large number of points.

The mapper algorithm is also included in the *giotto-tda* library [21], which implements visualization capabilities as widgets within the *Jupyter Notebook* environment [11]. Users can visualize the mapper graph in a static or an interactive mode. In the interactive mode, a *Jupyter Notebook* widget is used to modify some of the mapper parameters; although the interactivity enabled *w.r.t.* the mapper graph object is limited. *giotto-tda* uses Plotly, a wrapper over *D3.js* to provide some visualization capabilities. It focuses on creating a Pipeline object that interfaces with *scikit-learn* for downstream analysis (e.g., using the mapper graph for classification or parameter tuning via a grid search). Both *giotto-tda* and *Mapper Interactive* are equipped with on-the-fly computation of mapper graphs; however, the latter comes with a more scalable and extendable implementation. *Mapper Interactive* provides more opportunities to interact with the mapper graphs via data analysis and machine learning (ML) modules (such as applying linear regression to a subset of nodes in a mapper graph).

[1] https://mapperinteractive.github.io/

Gudhi [23] is a TDA toolkit that contains a version of the mapper algorithm. It defers the visualization of the mapper graph to other tools, such as *Graphviz* [6], *Geomview* [17], and *KeplerMapper*.

Comparison with the state-of-the-art. To further illustrate the capabilities of *Mapper Interactive*, we compare its features against two of the state-of-the-art tools, namely, *giotto-tda* (GT) and *KeplerMapper* (KM), as shown in Table 1. Features that are unique to *Mapper Interactive* include (but are not limited to): supporting interactive parameter adjustment via a visual interface; selecting nodes from a mapper graph that form connected components or paths for comparative analysis and ML tasks; easily-extendable visual interface via low-code development; and scalable GPU implementations for high-dimensional (100D+) point cloud data. In particular, *Mapper Interactive* supports path selection, as certain paths in the mapper graphs have been shown to be interesting in studying high-dimensional parameter space of plant phenomics [10]. Neither *KeplerMapper* nor *giotto-tda* supports the selection of paths or connected components of graphs for local, on-the-fly ML tasks such as dimensionality reduction or linear regression. Although *KeplerMapper* provides certain interactivity in exploring a pre-computed mapper graph (such as selecting and displaying details for a single node), it does not allow interactive exploration of multiple parameter combinations on-the-fly. *giotto-tda* recomputes mapper graphs based on parameters with a Python widget; however it has to regenerate HTML widgets for each dataset, and its mapper graph layout is static. On the other hand, *giotto-tda* and *KeplerMapper* have their unique features. Both *giotto-tda* and *KeplerMapper* can be run as libraries inside another Python script. *giotto-tda* outputs a Pipeline object that interfaces naturally with other *scikit-learn* objects.

Features	MI	GT	KM
Mapper graph computation and visualization			
Visualize a pre-computed mapper graph	Y	N	Y
Dynamic mapper graph layout	Y	N	Y
Compute mapper graphs via command line API	Y	N	N
Compute mapper graphs via a standalone GUI	Y	N	N
Update colormaps for continuous variables	Y	Y	Y
Enable glyphs for categorical variables	Y	N	N
Dynamic node size adjustment	Y	Y	N
Node contraction and edge filtering in GUI	N	Y	N
Node selection			
Select and display details for a single node	Y	Y	Y
Select and display labels for a subset of nodes	Y	N	N
Select nodes from a component for analysis	Y	N	N
Select nodes along a particular path for analysis	Y	N	N
Data analysis and machine learning (ML)			
Apply ML to the entire mapper graph	Y	Y	N
Apply ML to a selected subset of nodes	Y	N	N
Compare nodes with regression	Y	N	N
Compare mapper results with other ML results	Y	Y	Y
Parameter control			
Adjust parameters via a standalone GUI	Y	N	N
Adjust parameters in a *Jupyter Notebook*	N	Y	N
Changing clustering approaches via *scikit-learn*	Y	Y	Y
Change filter function	Y	Y	Y
Implementation and I/O			
Easily extensible GUI (low-code development)	Y	N	A
GPU implementation for mapper computation	Y	N	N
Run as a library inside Python scripts	N	Y	Y
Provides *scikit-learn* Pipeline object	N	Y	N
Caching of intermediate steps with Pipeline	N	Y	N

Table 1: Comparing features of Mapper Interactive (MI) against giotto-TDA (GT), and Kepler Mapper (KM) respectively. Blue means "yes" (Y), pink means "no" (N), purple means "almost yes" (A).

3 Background

We review the mapper construction introduced by Singh *et al.* [19]. *Mapper Interactive* visualizes the 1D skeleton of a mapper construction, referred to as the *mapper graph*, which provides a "skeleton-like" topological summary of a high-dimensional point cloud.

Given a high-dimensional point cloud $\mathbb{X} \subset \mathbb{R}^d$, we construct the *nerve of a covering*. A *cover* of $\mathbb{X}$ is defined as a set of open sets in $\mathbb{R}^d$, $\mathcal{U} = \{U_i\}_{i \in I}$ such that $\mathbb{X} \subset \cup_{i \in I} U_i$ (I being the index set). The 1D nerve of $\mathcal{U}$, denoted as $\mathcal{N}_1(\mathcal{U})$, is a graph. Each node $i \in I$ in $\mathcal{N}_1(\mathcal{U})$ represents a cover element U_i, and there is an edge between $i, j \in I$ if $U_i \cap U_j$ is nonempty. Fig. 1a gives an example in which $\mathbb{X}$ is a 2D point cloud sampled from the silhouette of a snowman. The cover $\mathcal{U}$ of $\mathbb{X}$ consists a collection of rectangles on the plane. The 1D nerve $\mathcal{N}_1(\mathcal{U})$ of $\mathcal{U}$ is the graph in Fig. 1c.

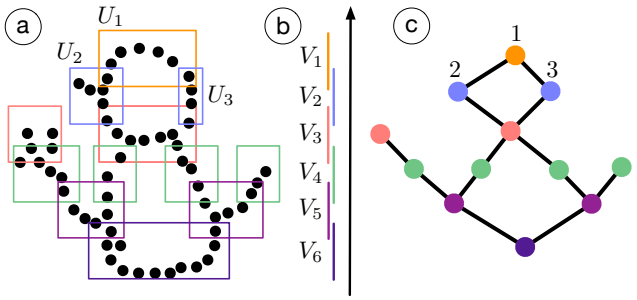

Figure 1: A mapper graph of a point cloud sampled from the silhouette of a snowman.

Given a point cloud $\mathbb{X}$, how does one obtain a cover of $\mathbb{X}$? In the classic mapper construction [19], obtaining a cover is guided by a set of scalar functions defined on $\mathbb{X}$. For simplicity, we work with a single scalar function f defined on $\mathbb{X}$, $f : \mathbb{X} \to \mathbb{R}$.

We start with a finite cover of a subset of the real line using intervals, that is, a cover $\mathcal{V} = \{V_k\}$ $(1 \le k \le n)$ of $f(\mathbb{X}) \subset \mathbb{R}$, such that $f(\mathbb{X}) \subseteq \cup_k V_k$; see Fig. 1b. We obtain a cover $\mathcal{U}$ of $\mathbb{X}$ by considering the clusters induced by points in $f^{-1}(V_k)$ for each V_k as cover elements. The 1D nerve of $\mathcal{U}$, denoted as $\mathcal{M} = \mathcal{M}(\mathbb{X}, f) := \mathcal{N}_1(\mathcal{U})$, is the *mapper graph* of $(\mathbb{X}, f)$.

Take Fig. 1 as an example: a point cloud $\mathbb{X}$ is equipped with a height function, $f : \mathbb{X} \to \mathbb{R}$. Six intervals form a cover $\mathcal{V} = \{V_1, V_2, \cdots, V_6\}$ of the image of f, that is, $f(\mathbb{X}) \subset \bigcup_k V_k$. For each k $(1 \le k \le 6)$, $f^{-1}(V_k)$ induces some clusters that are subsets of $\mathbb{X}$; such clusters form cover elements of $\mathbb{X}$. For instance, as illustrated in Fig. 1a, $f^{-1}(V_1)$ induces a single cluster of points that are enclosed by the orange cover element U_1, and $f^{-1}(V_2)$ induces two clusters of points enclosed by the blue cover elements U_2 and U_3. The mapper graph in Fig. 1c shows an edge between node 1 and node 2 since $U_1 \cap U_2 \neq \emptyset$. It captures the overall shape of the snowman.

Algorithmic details in practice. Given a point cloud $\mathbb{X}$, several parameters are needed to compute the mapper graph $\mathcal{M}$, including a function $f : \mathbb{X} \to \mathbb{R}$ (referred to as the *filter function*), the number of cover elements n and their percentage of overlaps p, the metric $d_{\mathbb{X}}$ on $\mathbb{X}$, and the clustering method. For instance, for the example in Fig. 1, f is the height function, $n = 6$ and $p = 30\%$, $d_{\mathbb{X}}$ is the Euclidean distance, and the clustering method is DBSCAN.

In practice, the choice of the filter function f is nontrivial. Common choices include the L_2-norm, variants of geodesic distances, and eccentricity [1, 19]. The mapper graph $\mathcal{M}(\mathbb{X}, f)$ captures the topological summary of the data $(\mathbb{X}, f)$, that is, $\mathbb{X}$ coupled with f; hence, a different choice of f gives rise to a different type of summary. Each interval (cover element) typically has uniform size. Some libraries (such as *giotto-tda*) offer a "balanced" cover where

the inverse image of each interval contains an equal number of points.

A common choice for the clustering method is DBSCAN [7], which is a density-based clustering algorithm. DBSCAN contains two parameters: ε is the neighborhood size of a given point, and *minPts* is the minimum number of points needed to consider a collection of points as a cluster.

The filter function f may be generalized to be a multivariate function, that is, $f : \mathbb{X} \to \mathbb{R}^m$ (for $m \geq 2$). In most practical scenarios, $m = 2$, and the resulting mapper graph is referred to as a *2D mapper graph*. The corresponding cover elements of $\mathbb{R}^2$ become rectangles. *Mapper Interactive* supports the computation of both 1D and 2D mapper graphs.

4 Design and Implementation

We discuss three main capabilities of *Mapper Interactive*: *interactive* user interface for on-the-fly computation and exploration of mapper graphs across a range of parameters; *extendable* visualization design for novice and expert users; and a command line API that provides *scalable* backend computation of mapper graphs.

4.1 Interactivity

The user interface of *Mapper Interactive* is shown in Fig. 2. It contains three main interactive panels: (a) the graph visualization panel, (b) the selection panel, and (c) the control panel.

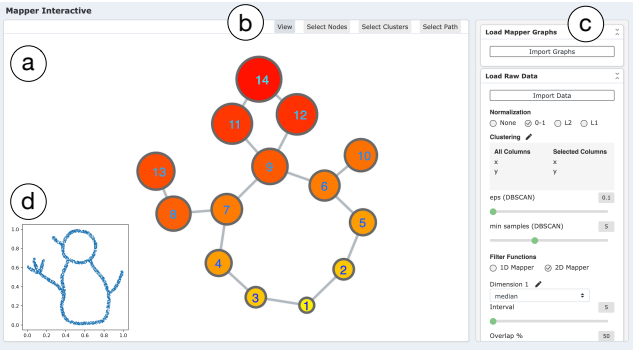

Figure 2: User interface of *Mapper Interactive*.

The **graph visualization panel (a)** visualizes the resulting mapper graph using a force-directed layout, which summarizes the underlying structure of an input point cloud dataset. It enables basic interactive operations such as zooming, dragging, and panning. In Fig. 2, we see an example of a mapper graph computed from the snowman point cloud (Fig. 2d) that appears in Fig. 1 of Sect. 3.

The **selection panel (b)** enables users to select a subset of mapper graph nodes (and their underlying data points) under three data selection modes. As illustrated in Fig. 3, under the **select nodes** mode (Fig. 3a), users can select any number of the nodes in the mapper graph. Under the **select clusters** mode (Fig. 3b), users can select connected components of the mapper graph. Under the **select paths** mode (Fig. 3c), users can specify the start and end point of a path in the mapper graph and select a shortest path between them (if one exits). The path can also be extended by selecting another ending point (Fig. 3d). After selection, various analysis modules can be applied to the selected data points, including linear regression and dimensionality reduction, to study the properties associated with the selected data.

The **control panel (c)** provides parameter controls for computing mapper graphs on the fly. It includes data wrangling via the visual interface in addition to data wrangling provided via the command line API. When loading a point cloud dataset, the input data can be preprocessed through different normalization schemes such as min-max and L_2 normalization. Either 1D or 2D mapper graphs can be constructed, depending on the number of filter functions. A filter function can be specified based on a chosen dimension (column) of the input data, or based on derived properties from the point clouds such as L_2-norm, density, and eccentricity [19]. For clustering, the default approach is DBSCAN. Agglomerative clustering and mean shift clustering are also available through the user interface and command line tool. As the parameters change, the resulting mapper graph can be computed on the fly. In addition, the control panel interfaces with precomputed mapper graphs obtained from the command line API.

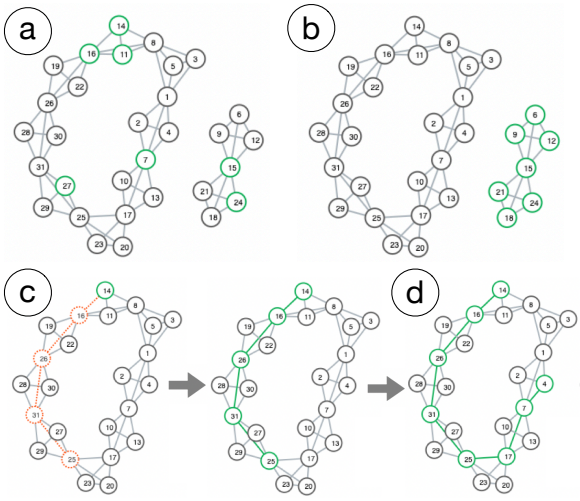

Figure 3: Three data selection modes for the mapper graph: (a) **select nodes**, (b) **select clusters**, and (c) **select paths** that include (d) path extensions.

The control panel also specifies parameters associated with the visual encoding of the mapper graph. Nodes can be colored according to a chosen dimension (variable/column) of the input data, or by the number of points contained in them. For discrete variables, a pie chart that reflects the composition of each node is drawn on top of each node. For continuous variables, a continuous colormap is applied, with user-specified color encodings and range of values. The size of the nodes can be adjusted using the value of a chosen variable or the number of points in the cluster (see Fig. 4). When a subset of nodes is selected, the control panel displays the details of each node by drawing a bar chart of average values for numerical columns, and displaying the individual information of points contained in each node cluster (see Fig. 5).

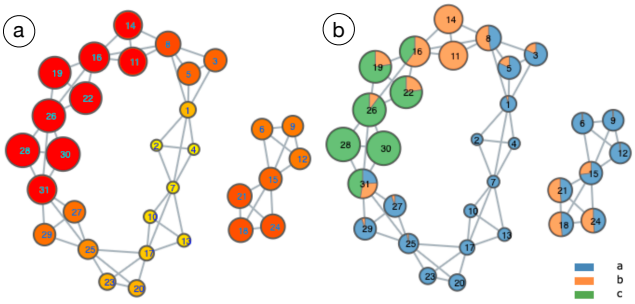

Figure 4: An example of the nodes colored by the average x coordinates (a) and the point labels (b). The size of each node is adjusted using its average x coordinate.

The control panel also provides data analysis and machine learning modules for users to better understand the results of the mapper algorithm, which is currently not possible with other existing tools. Machine learning techniques, including linear regression and princi-

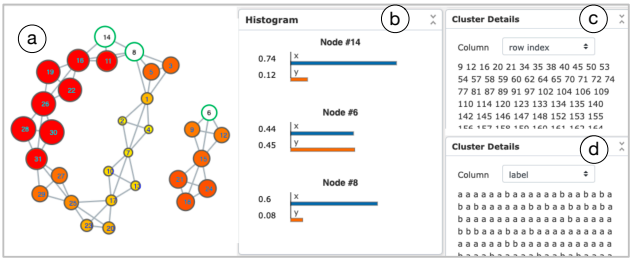

Figure 5: (a) In the mapper graph, nodes 6, 8, 14 are selected. (b) The bar chart of average values of x and y coordinates for each selected node. (c) The row indices of the union of all points contained in the selected nodes. (d) The labels of the union of all points contained in the selected nodes.

pal component analysis (PCA), can be applied to analyze a selected subset of nodes. If no selected nodes are available, the entire dataset will be taken as input. Take Fig. 6 for example: (a) shows a 3D point cloud sampled from the model of a horse; (b) is the 2D PCA result with k-means clustering applied to the projected data, where colors represent different clusters; (c) is the mapper graph of (a) generated by *Mapper Interactive* with nodes 1, 7, 13, 19 at the four feet of the horse, node 8 at its tail, and node 34 at its head; and (d) shows the results of applying linear regression to the point cloud, where x and z are the independent variables, and y is the dependent variable.

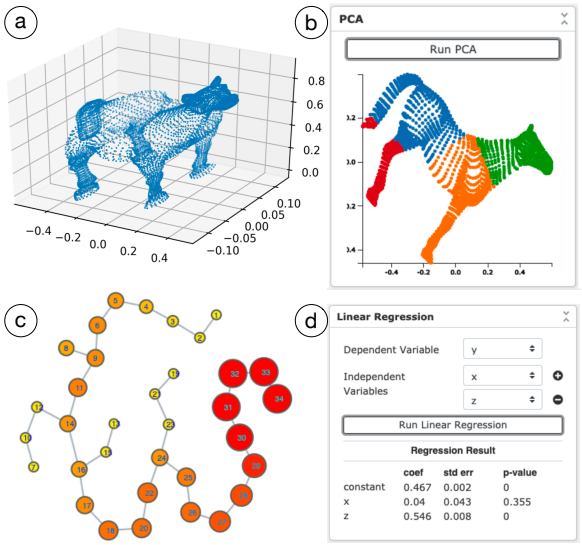

Figure 6: (a) A three-dimensional (3D) point cloud sampled from a model of a horse. (b) the 2D PCA result combined with k-means clustering where $k = 4$. (c) The resulting mapper graph. (d) Linear regression result of regressing y on x and z.

4.2 Extendability

Mapper Interactive allows users to easily extend the framework by adding new data analysis and visualization modules to the control panel, primarily via interfacing with Python's *scikit-learn* package. Such extendability brings flexibility for users to apply machine learning techniques to nodes (clusters) of interest that arise from the mapper graph. It also enables users to explore the properties associated with these nodes.

We provide two modes for different user groups to extend the framework: the novice user mode and the expert user mode.

Novice user mode. For users with limited programming experience, we provide an easy way for them to add new modules. All they need to do is to describe the new module information within the new_modules.json file, and the system will detect and generate all the new modules inside that json file automatically. Currently, *Mapper Interactive* allows the addition of supervised and unsupervised learning algorithms that are available via *scikit-learn*. For each new module, users need to specify the function name, function parameters, and whether it is a supervised or unsupervised model for the Python backend to fit the model correctly, along with a list of visual component types for the JavaScript frontend to visualize the result. For a supervised learning module, users need to provide additional information about the independent and dependent variables. For visualization purpose, we provide commonly used visual components, such as scatter plots, line graphs, and tables, to be integrated with the result of a new module.

a
```
{"modules":
    [
        {
            "name": "TSNE",
            "function-name": "sklearn.manifold.TSNE",
            "function-parameters":{"n_components":2},
            "module-type": "unsupervised_learning",
            "components": ["scatter plot"]
        }
    ]
}
```

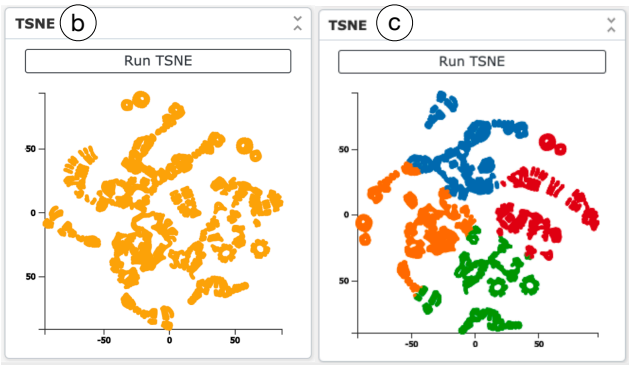

Figure 7: An example of adding t-SNE module to the control panel of *Mapper Interactive*.

Expert user mode. For expert users with programming experience, we provide a template function call_module_function in Python within *Mapper Interactive*. It supports customizable and multistep analysis pipelines. We also provide a template class in JavaScript for creating new visual components using *D3.js*. With a few lines of code, users can add a new drawing method within the template class to modify a visual encoding. The styles of visual components are changed via a CSS file.

We give an example in Fig. 7 for adding a t-SNE module to the *Mapper Interactive* interface. t-SNE is a nonlinear dimensionality reduction technique that is quite popular in practice. In (a), under the novice user mode, the information of a new module that performs t-SNE-based dimensionality reduction is added to the new_modules.json file. With less than six lines of code, the t-SNE module is now part of the *Mapper Interactive* interface where a 3D point cloud from a horse is visualized in 2D (b). In (c), under the expert user mode, by adding a few lines of code to the Python script call_module_function, the t-SNE result can be further enhanced using k-means clustering (where $k = 4$).

4.3 Scalability

Mapper Interactive is equipped with scalable backend computation of mapper graphs. In particular, it comes with a command line API for data wrangling and the computation of mapper graphs. The single processor CPU implementation of the mapper algorithm in *Mapper Interactive* is 3 to 6 times faster than its vanilla implemen-

tation. We also provide a GPU implementation that provides an additional 2-fold speedup for 1 million points in 128 dimension.

Key implementational idea. We present a simple but effective strategy for speeding up *any* mapper algorithm framework that uses DBSCAN as a subroutine.

The backend mapper implementation of *Mapper Interactive* is built upon *KeplerMapper*. *KeplerMapper* is a user-friendly implementation of the mapper algorithm that provides some interactive capabilities. However, its default mapper graph computation (considered as the vanilla implementation) does not scale well with the size of the point cloud. The computational bottleneck happens during the DBSCAN clustering stage in which the algorithm queries all pairwise distances. The first idea is parallelizing individual clustering instances, that is, computing the clusters for the inverse map of each interval. This strategy provides some amount of speed-up, which is employed by both *Mapper Interactive* and *giotto-tda*.

In *Mapper Interactive*, we push the parallelization even further. We modify the algorithm by precomputing the distance matrix of points within each interval using *scikit-learn*'s highly optimized `pairwise_distance` function. This function converts the distance computations in the clustering algorithm to a lookup in the precomputed matrix, achieving significant speed-up at the cost of higher memory usage entailed by storing the precomputed distances. For *Mapper Interactive*, such a strategy is shown to be 6 times faster than the vanilla implementation for 300K points for *ImageNet* dataset.

In fact, the strategy employed by *Mapper Interactive* is applicable to *any* mapper framework employing DBSCAN as a clustering subroutine. By enforcing pre-computation of distance matrices as a user-specified parameter in DBSCAN, our strategy also helps to speed up mapper graph computation for both *giotto-tda* and *KeplerMapper*, when the point clouds are of significantly high dimension (100D+). It is also important to point out that precomputing distances ceases to be effective when the pullback cover sets become too large. It also does not lead to significant speed-up when the dimensionality is not sufficiently large.

In the experiments below, we perform runtime analysis for *Mapper Interactive*, *KeplerMapper* (version 1.2.0) and *giotto-tda* (version 0.3.1), where GT and KM represent their default configurations, respectively. To demonstrate that our strategy will speed up any DBSCAN-based mapper framework, we give performance numbers for GT* and KM*, which represent the improved configurations using the strategy of *Mapper Interactive* in precomputing distances, respectively[2].

Datasets. The *ImageNet* and *Cifar* datasets are created by passing input images to *InceptionV1* and *ResNet-18* neural networks respectively and collecting the activation vectors at an intermediate layer. The *ImageNet* dataset has 512 dimensions and 300K points, while the *Cifar* dataset has 256 dimensions and 3 million points. For the *Random* vector datasets, we sample 10 million points of 128 dimensions; each dimension is drawn from a uniform distribution over $[0, 1]$. For our experiments, we subsample each of the datasets at various order of magnitudes to demonstrate run time performance of each method at different input sizes.

Runtime analysis with the command line API. We first show comparisons of peak memory usage among *Mapper Interactive* (MI), GT, KM, GT*, and KM* in Fig. 8, the numerical values can be found in Table 2 of the supplement. Roughly speaking, for all three datasets, *Mapper Interactive* has up to 1.7× increase in peak memory usage compares to its vanilla implementation. Using our strategy, GT* has up to 4.0× memory increase over GT; and KM* has up to 1.9× memory increase over KM.

The runtime comparison is shown in Fig. 9(a-c) and Table 3 in the supplement. By employing space-time tradeoff of MI via the pre-computation of distance matrices, both GT and KM can

[2]Via detailed discussions with a giotto-tda developer.

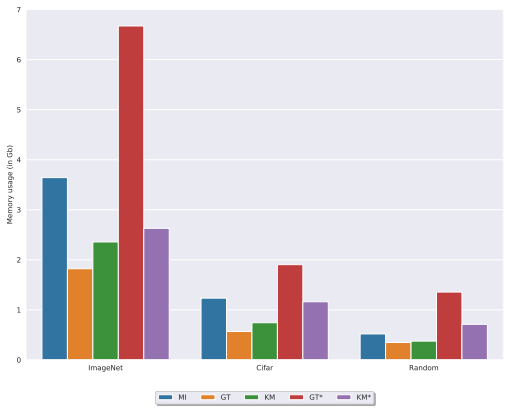

Figure 8: Peak memory usage (in Gigabytes) on three datasets, each with 100K points. The number of intervals is set to 100. The ImageNet, CIFAR, and Random datasets contain 512, 256, and 128 dimensional point clouds respectively.

be improved to achieve comparable performance with MI. For the *ImageNet* dataset of 300K points, MI achieves an approximately 6× speedup against its vanilla implementation (KM); with our strategy, GT* gets 9× speedup against GT, and KM* achieves 7× speedup *w.r.t.* to KM. For the *Cifar* dataset of 3 million points, MI achieve 4× speed-up against its vanilla implementation; while GT* gets 3× speed-up against *GT*, and KM* obtains 5× against KM. For the *Random* dataset of 10 million points, our command line API computes a mapper graph in 29 minutes, obtaining a 3× speed-up against its vanilla implementation; while KM* achieves 2× speed up vs KM, and GT* and GT run out of memory.

We perform the above experiments on an Intel Xeon 2.4GHz CPU with 16 cores and 32 GB RAM. For parallel computations, we restrict the methods to 8 cores to minimize effects from OS processes. We set the `n_jobs` parameters for the clustering algorithm in *KeplerMapper* to 8 and the same parameter for the *giotto-tda* implementation to 8 as well, to compare against the parallelized versions of our implementations.

Runtime analysis with the visual interface. Additional I/O and memory overhead is associated with computing the mapper graph via the visual interface in comparison with the command line API.

To test the scalability of the visual interface, we use a macOS system on an Intel 2.3 GHz Core i5 CPU and 8 GB RAM. For a dataset with 100K points in 512 dimensions, it takes an average 3 minutes to compute and render the mapper graph in the browser. When the number of points increases to 200K, the computation and renders takes an average of 1 hour. On the other hand, if we generate the mapper graphs with 1 million points using the command line API, the interface can easily load the resulting mapper graphs in under 1 minute. By interfacing with the command line API, we are able to explore larger point cloud data via the visual interface with precomputed mapper graphs.

GPU accelerated distance computation. Finally, in order to further speed up our parallel mapper algorithm, we introduce a GPU-based distance computation. We use *PyTorch* to accelerate the distance computation, moving away from *scikit-learn*. Our results are based on a computer with a 32-core Intel Xeon CPU (1.8 GHz), 132 Gb of RAM, and a Nvidia Titan V GPU with CUDA 10.1. We notice a roughly 2× speed-up in comparison with our CPU implementation for our larger point clouds, see Fig. 9(c-f) (numerical values are shown in Table 4 of the supplement). In particular, for 1 millions points, our GPU implementation achieves a 1.6×, and 2.1× speed-up for the *Cifar* and the *Random* dataset, respectively.

In summary, the backend GPU implementation of mapper graph computation achieves (on average) between 6× to 12× speed-up in comparison with the CPU-based vanilla implementation. However,

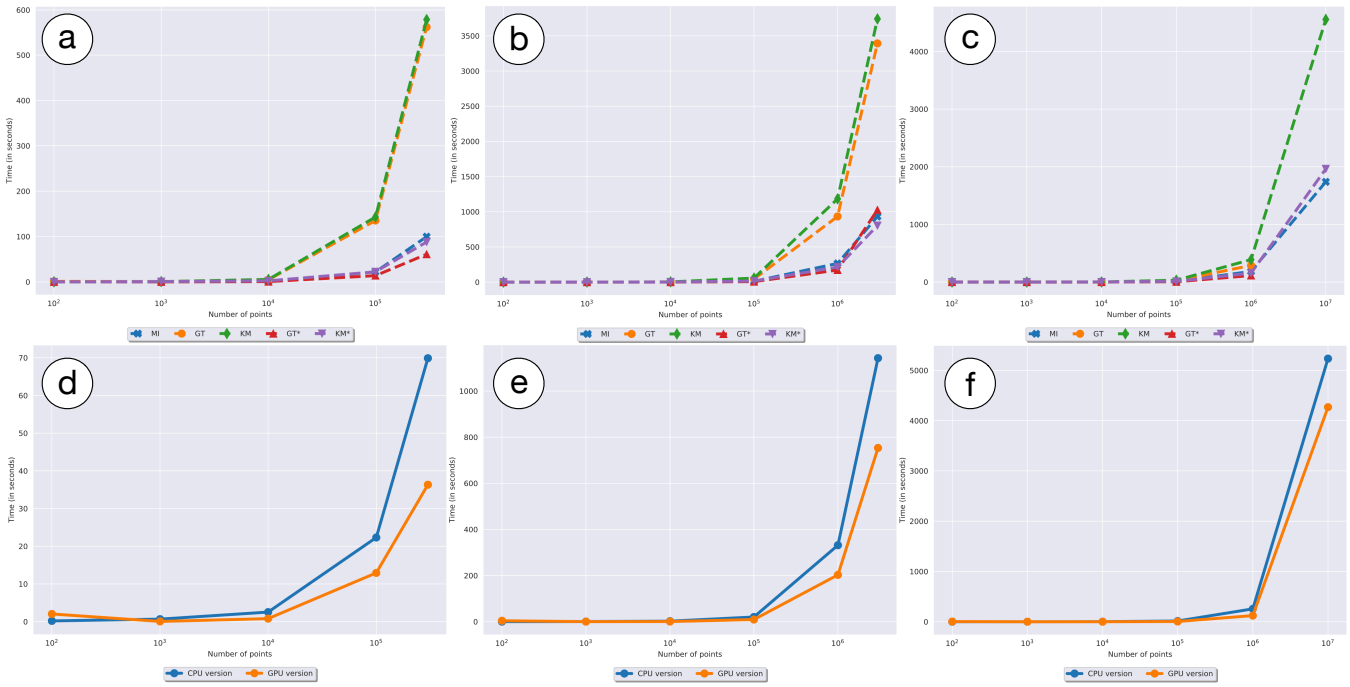

Figure 9: Top row (a-c): CPU runtime (in seconds) on the *ImageNet*, *Cifar*, and *Random* datasets, respectively. Bottom row (e-f): GPU runtime (in seconds) in comparison with CPU runtime for our implementation. Labels are shown on the x-axis using a log scale.

a communication overhead is incurred when large arrays of data are moved from CPU to GPU and back. As a result, experiments with a higher number of intervals (500+) do not provide as large of a speed-up.

4.4 Implementation

The visual interface of *Mapper Interactive* is implemented using HTML/CSS/JavaScript stack with *D3.js* and *JQuery* JavaScript libraries. It interfaces with a Python backend via a *Flask* server. The mapper graph computation is modified from the *KeplerMapper* implementation. Python libraries, including *scikit-learn*, *statsmodels*, and *scipy*, are used for its machine learning modules.

We also provide a Python command line API which is designed for data wrangling and offline mapper graph computations for large datasets. The wrangling process handles missing values, identifies numerical and categorical columns, and removes non-numerical elements from the numerical columns. The wrangled data may be imported to the visual interface for interactive exploration. To compute mapper graphs via the API, users can specify the range of parameters for the mapper algorithm, including the number of intervals, the amount of overlap, the number of threads to use when computing pairwise distances, parameters for DBSCAN, etc. Users can also specify GPU acceleration for computation via the command line API. The resulting mapper graphs are put into a single folder to be interfaced with the visual interface for interactive exploration.

5 Use Cases

We demonstrate the utility of *Mapper Interactive* via three use cases on well-known and new datasets from image classifiers, breast cancer, and COVID-19. By applying *Mapper Interactive* to these datasets, we showcase the usability, interactivity and extensibility of the tool. While a subset of the findings in these use cases may be obtained using existing frameworks, the main advantage of *Mapper Interactive* is that it provides the largest set of features unavailable with existing frameworks (cf. Table 1), and it makes mapper algorithms accessible to nonspecialists (with little background in Python and TDA) via a low-code development environment. In addition, with zero or a few lines of code, *Mapper Interactive* enables a quick way to check if TDA is a viable tool for a given application.

Through these use cases, we demonstrate that *Mapper Interactive* is essential in providing fast and easy ways to prototype and experiment with user-specified datasets, thus helping accelerate TDA workflows for fast insight generation.

5.1 Discovering the Divergence of COVID-19 Trends

Our first use case is to analyze and compare COVID-19 trends in the United States. The key point is that *Mapper Interactive* enables fast insight generation on a new dataset.

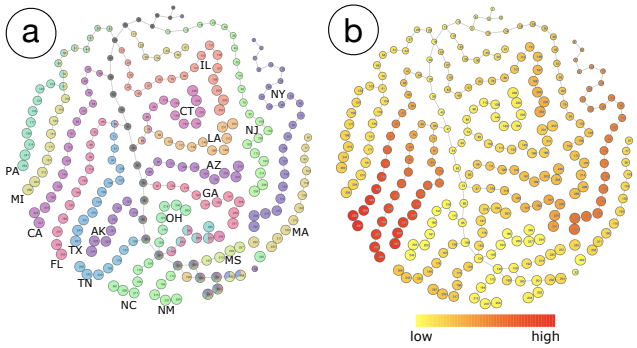

Figure 10: The mapper graph of the full COVID-19 dataset. The graph nodes are colored by the composition of the states (a) and the number of confirmed cases (b), respectively. For DBSCAN, we chose $\varepsilon = 0.1$, $minPts = 5$. For mapper graph, we set $n = 35$ and $p = 50\%$. Each dimension is normalized by a min-max scale. The size of nodes indicates the average number of recorded days.

The dataset contains the daily records of COVID-19 cases in all 50 states from April 12, 2020 to September 18, 2020[3]. It contains 9240 data points (rows), each of which corresponds to a daily record for a given state. For each state, it contains 7 statistical measures (columns): number of confirmed cases, death cases, active cases,

[3]https://github.com/CSSEGISandData/COVID-19/

people tested, as well as the testing rate, mortality rate, and incidence rate (i.e., the number of cases per 100K persons).

We first compute an initial mapper graph using all data points. We include all 7 dimensions (columns) to compute the pairwise distance matrix, and use the number of recorded days (since April 12, 2020) as the filter function. The number of recorded days indicates how many days have passed from the record starting date (April 12, 2020) to the date associated with each row of data.

The result is shown in Fig. 10. Certain states are shown to be separated from other states and form their own connected components, such as New York (NY) and Massachusetts (MA), which implies that their statistics (and thus "epidemic trends") may be quite different from others. To further investigate why and how these states are separated from one another, we select nine states (AZ, CA, FL, GA, IL, NC, NJ, NY, TX) with the largest number of confirmed cases and compute a second mapper graph.

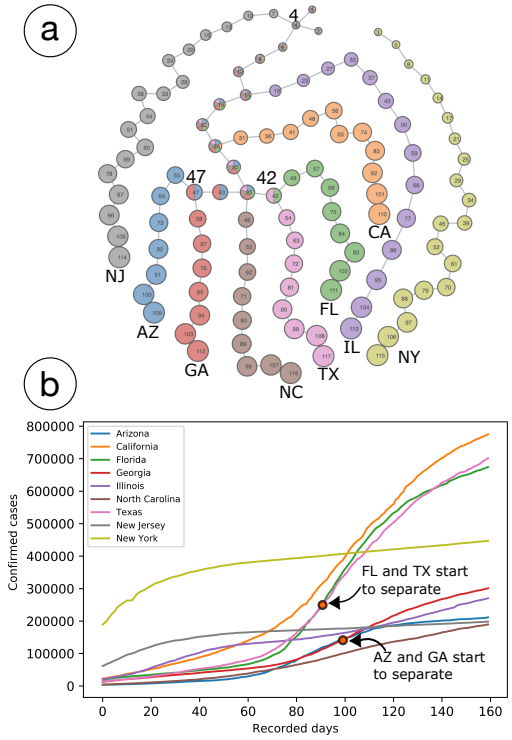

Figure 11: (a) The mapper graph for the selected states. (b) The line graph of the daily confirmed cases; the x-axis represents the number of recorded days, and the y-axis represents the confirmed cases. For DBSCAN, $\varepsilon = 0.15$, $minPts = 5$. For the mapper graph: $n = 20$, $p = 50\%$. Each dimension is normalized by a min-max scale.

As shown in Fig. 11a, the states become separated from each other after certain branching points. The size of each node is encoded by the average number of recorded days. By comparing the line graph of the confirmed cases (Fig. 11b), we can see that the order by which each state is separated is related to how different its curve is from that of other states.

For example, as shown in the line graph in Fig. 11b, the curve of New York (NY) deviated the most from other states, so in the mapper graph (Fig. 11a) it is not connected with any other state, thus forming its own connected component. New Jersey (NJ) shows the second highest deviation besides New York in the line graph, so it splits from the main branch at node 4 in the mapper graph.

Arizona (AZ) and Georgia (GA), as well as Florida (FL) and Texas (TX), are two pairs with similar trends in Fig. 11b, so their nodes are the last ones to be separated from the main branch in Fig. 11a. In particular, the average number of days at node 47 is 100, which reflects exactly where Arizona and Georgia start to separate in the line graph. The average number of days at node 42 is 92, which reflects when Florida and Texas start to separate in the line graph.

Therefore, through the resulting mapper graph, we are able to distinguish states with different epidemic trends and to determine how different their trends are. We can also discover when their trends start to diverge by looking at the nodes at the branching points.

5.2 Visualizing Class Separation via Neuron Activations

Our second use case is to visualize neuron activations collected at the last layer of an image classifier to study the degree to which classes are separated during training. The key point is that *Mapper Interactive* helps to highlight class separation with a categorical dataset, and it can be extended to perform in-depth analysis of the data.

The *Cifar* dataset is created by passing input images from CIFAR-10 [12] to *ResNet-18* neural network, and collecting activation vectors (that is, combinations of neuron firings) from the last layer (referred to as "4.1.bn2") of the network. We then treat these activation vectors as a dataset containing high-dimensional points and apply *Mapper Interactive* to it. We use $50K$ images from 10 image classes, namely ship, truck, automobile, horse, deer, bird, dog, cat, fog, and airplane. Each image corresponds to an activation vector with 512 dimensions.

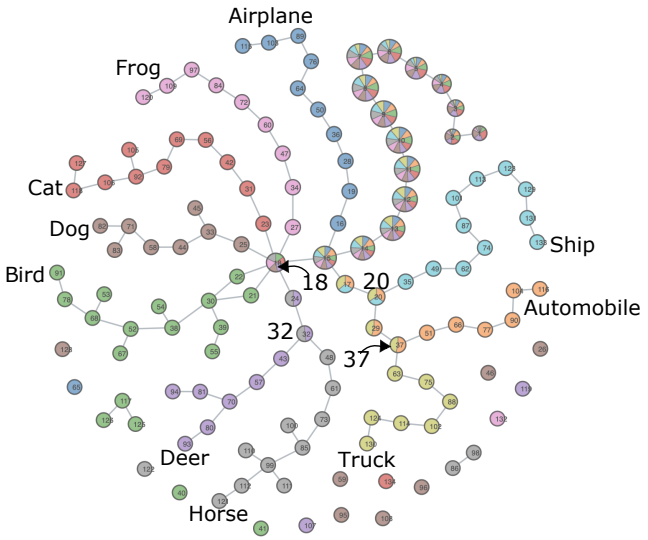

Figure 12: The mapper graph of the activation vectors. For DBSCAN, $\varepsilon = 8.71$, $minPts = 5$. For mapper graphs, $n = 40$, $p = 0.2$.

We compute the pairwise distance matrix using all 512 dimensions, and use L_2-norm as the filter function. As shown in Fig. 12, the resulting mapper graph highlights the separation of image classes at the last layer of a trained neural network (ResNet-18) with high classification accuracy. By drawing a pie chart on top of each node, the proportions of categories within each node are clear. The size of each node reflects the number of points within the node (cluster).

The resulting mapper graph not only clusters images from each class into a separate branch, but also highlights the relationship among the different classes. For example, *Mapper Interactive* highlights the observation from [18]. A branch of nodes containing the deer and horse images first emerged from the branching node 18, which contains images from several classes. Then, the two classes are separated into two branches at node 32. The branching order indicates that the deer and horse images are more similar than images from other class categories. Similarly for the automobile and truck images, a branch containing images from both categories first emerged from branching node 20, and then the two categories

were separated from each other at branching node 37, indicating the automobile and truck images are more similar than other images.

Using *Mapper Interactive*, we can easily perform additional analysis to further advance our understanding of the dataset. We can extend the tool by adding a PCA analysis module and a module for visualizing the distribution of nearest neighbor distances. Fig. 13a applies PCA to the activation vectors. The colors correspond to the 10 class categories used in the mapper graph. Compared to the mapper graph, the PCA projection of the activation vectors does not separate the classes well, and the relationship between different classes is not well depicted.

In addition, for such a large dataset, parameter tuning can be difficult and time consuming. We demonstrate how to add a new module for tuning the ε parameter in DBSCAN by creating a module in the expert mode. We use the Python library *PyNNDescent* to approximate a nearest neighbor search for the point cloud data, sort the distances of the k-th nearest neighbor, and plot the distance distribution to figure out the most appropriate ε value.

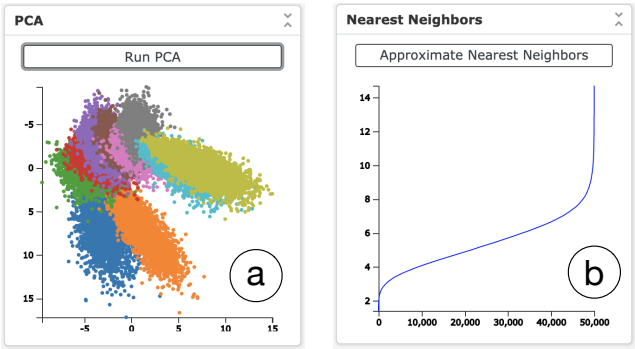

Figure 13: Adding two additional analysis modules. (a) The result of a PCA module. (b) Computing the distance distribution of the fifth nearest neighbors for all points.

Our objective is to have, with the right ε value, a maximum number of points with at least k neighbors, with the maximum distance not too large to include too many neighbors. For the CIFAR-10 dataset, we chose $k = 5$, and the distance distribution of the fifth nearest neighbors is shown in Fig. 13b. The line plot shows that an ε value around 8 is most appropriate. In DBSCAN clustering, the parameter ε is the maximum distance between two points for them to be considered as neighbors. If ε is too small, the number of neighbors for most points will be less than the minimum number of points to be clustered together, and most points will be considered as noise, resulting in a suboptimal clustering. If ε is too large, then all the points will be considered as neighbors of each other, and thus be assigned to the same cluster, which will also be suboptimal. The nearest neighbor module in Fig. 13b, therefore, helps the users choose the optimal parameter for DBSCAN.

5.3 Exploring Breast Cancer Data

Our third use case is to provide alternative ways to explore the results from a breast cancer study [13]. Lum *et al.* [13] utilized the mapper algorithm to identify subgroups in breast cancer patients. Using *Mapper Interactive*, the key point is that we can consider alternative configurations of the mapper algorithm to explore these breast cancer datasets and obtain similar insights. Due to the extendability of the tool, we can further provide in-depth regression analysis to identify possible factors that are highly correlated with patient survival.

Insight discovery. We first discuss insight discovery with alternative configurations of the mapper algorithm. We use the two breast cancer datasets studied by Lum *et al.* [13], referred to as the *NKI* dataset [25], and the *GSE2034* dataset [26].

The *NKI* dataset contains information from 272 breast cancer patients (rows). For each patient, two types of variables (columns) are

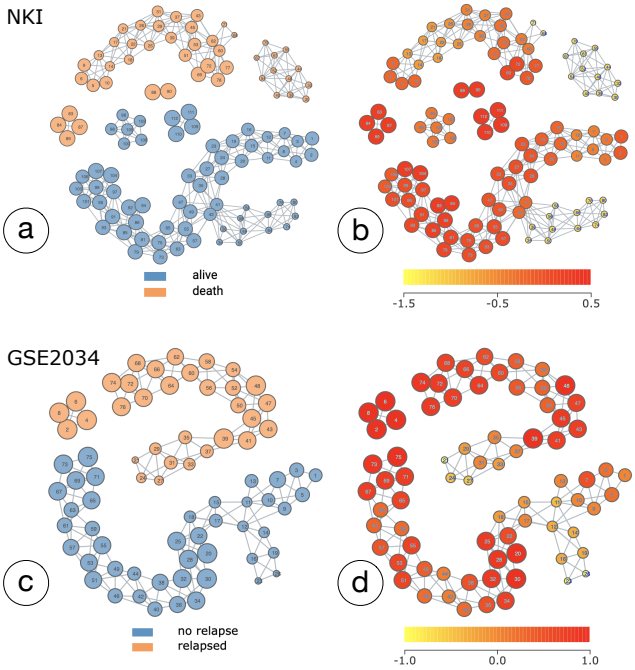

Figure 14: (a-b) NKI mapper graphs. For DBSCAN, $\varepsilon = 15$, $minPts = 2$. For mapper graph: $f_1 = L_\infty$-norm, $n_1 = 78$, $p_1 = 65\%$; $f_2 = event\_death$, $n_2 = 10$, $p_2 = 68\%$. (c-d) GSE2034 mapper graphs. For DBSCAN, $\varepsilon = 0.45$, $minPts = 2$. For mapper graph: $f_1 = L_\infty$-norm, $n_1 = 37$, $p_1 = 72\%$; $f_2 = relapse$, $n_2 = 5$, $p_2 = 50\%$. Each dimension is normalized by a min-max scale.

recorded: the first type of variables contains 1554 gene expression levels, and the second type of variables consists of other medical records or physiological measures. The physiological measure columns include *event_death* (whether a patient survived or not), *survival_time*, *recurrence_time*, *chemo* (whether a patient received a chemotherapy), *hormonal* (whether a patient received hormonal therapy), *amputation* (whether forequarter amputation has been used), *hist_type* (histological type), *diam* (diameter of the tumor), *posnodes* (number of nodes), *grade* (cancel level), *angio_inv* (to what degree the cancer invaded blood vessels and lymph vessels), and *lymph_infil* (level of lymphocytic infiltration).

To compute the mapper graph of the *NKI* dataset, we inherit some parameter configurations from [13], with the exception that we use DBSCAN as our clustering algorithm instead of the single-linkage clustering employed by Lum *et al.* in [13]. Subsequently, we use slightly different parameters for the number of intervals n and the amount of overlap p that is adaptive to DBSCAN. We take the 1500 mostly varying genes to form a point cloud in 1500 dimensions, compute their Euclidean pairwise distance matrix, and construct a 2D mapper graph using L_∞-norm and the response variable *event_death* as its filter functions.

The *GSE2034* dataset, on the other hand, consists of gene expression levels of 22283 genes from 286 patients. Instead of recording the survival data, this dataset provides a variable *relapse* to indicate whether the patient suffered a relapse. We take the top 10 most varying genes to compute the pairwise distance matrix, and construct a 2D mapper graph using L_∞-norm and the *relapse* variable as its filter functions.

Lim *et al.* [13] used mapper graphs to study subgroups of breast cancer patients. In most cases, the expression level of the estrogen receptor gene (ESR1) is positively correlated with the prognosis. Patients with high ESR1 levels usually have a better prognosis and are more likely to survive than patients with low ESR1 levels. How-

ever, among all the patients with high ESR1 levels are subgroups having poor clinical outcomes. Patients who had low ESR1 levels but survived were also identified over the years. Researchers have studied such subgroups using certain experimental data [16, 20, 22]. However, the challenge is to identify subgroups under more general settings, such as data from different sets of patients that are collected at different times.

We therefore apply *Mapper Interactive* to both *NKI* and *GSE2034* datasets using slightly different parameter configurations in comparison to [13]. The resulting mapper graphs are shown in Fig. 14. It is interesting to observe that the two resulting mapper graphs consist of similar structure for the survivor/non-relapse patients; and they share similar (but not identical) structures in comparison to the results from [13]. In each graph, the blue connected component on the bottom contains a branch of nodes with low average ESR1 expression levels, thus defining a subgroup of survivor/nonrelapse patients. The result shows that we are able to visually identify similar subgroup structures under two datasets generated from totally different experimental settings.

In-depth exploration. Furthermore, we can easily extend *Mapper Interactive* to perform in-depth analysis of the breast cancer datasets (not discussed in [13]).

Since the *NKI* dataset contains medical records and physiological measures information about the patients, we can make use of the analysis modules to explore interesting subsets of the mapper graph. As illustrated in Fig. 15a, we first consider the clusters of the largest connected component among the survivors (the green selected clusters). Since there is a subgroup of low ESR1 patients in these clusters, we can easily apply the existing linear regression module to these clusters. Specifically, we are interested in understanding what variables have statistically significant effects on the expression levels of ESR1 without affecting the patient survival.

The result is shown in Fig. 15b. Under the significant level (p-value) of 0.05, the variables *amputation*, *grade*, and *lymph_infil* are significantly correlated with the expression levels of ESR1. Recall that the *amputation* variable indicates whether the patient has received the forequarter amputation treatment, the *grade* variable indicates the stage of the cancer, and the *lymph_infil* variable indicates the level of lymphocytic infiltration.

We explore which genes affect the survival of patients with low levels of ESR1. Since *event_death* is a binary variable, we add a new module to perform logistic regression. We include the top 10 genes that are selected using a recursive feature elimination. The result is shown in Fig. 15d. Under the p-value of 0.1, AL049963 is the only gene that is significantly correlated with the *event_death* variable. Further analysis based on these regression results would be an interesting avenue to explore in a follow-up study.

A subset of results for these use cases cannot be easily reproduced using other frameworks. *Mapper Interactive* supports the application of ML techniques to a subset of nodes interactively. Therefore, in the breast cancer example, we can easily apply regression to the identified subgroups using path-based and/or component-based node selection. *giotto-tda* or *KeplerMapper* will require more coding effort to achieve the same result. Neither *giotto-tda* nor *KeplerMapper* support pie charts on top of the nodes for discrete variables; therefore, it is hard to identify the branching points in both COVID-19 and the neuron activation examples. Overall, *Mapper Interactive* provides more interactivity and flexibility, and is less time intensive for nonspecialists exploring high-dimensional data with TDA.

6 Conclusion and Discussion

In this paper, we present *Mapper Interactive*, an interactive, extendable, and scalable toolbox for the visual exploration of high-dimensional data using the mapper graph. It supports computation and interactive exploration of mapper graphs. It is easily extendable, where both novice and expert users can add machine learning and

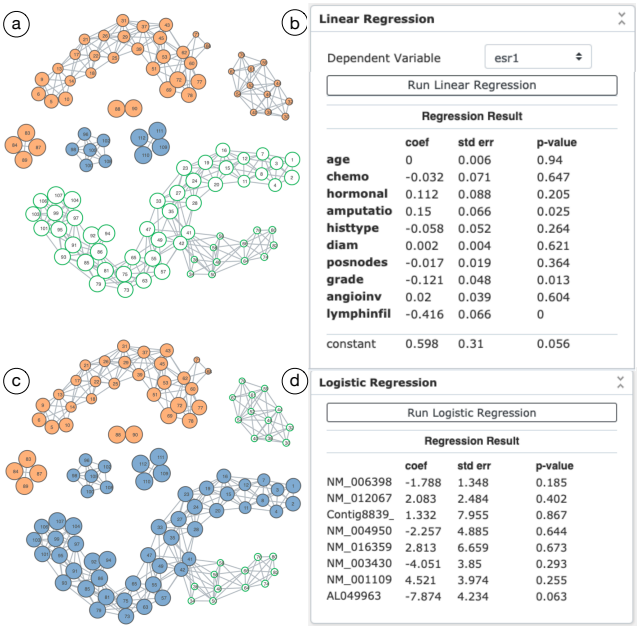

Figure 15: In-depth exploration of the *NKI* dataset. (a) White nodes with green edges are selected to perform linear regression on ESR1 levels. (b) Linear regression result. (c) White nodes with green edges are selected to perform logistic regression on *event_death*. (d) Logistic regression result.

visualization modules with a few changes to a json file or with a few lines of codes. Since it provides low-code development environment, it also enables a quick way to check if TDA is a viable tool for a given application. Its command line API can compute a mapper graph of 1 million points in 256 dimension in less than 3 minutes, which is roughly 4 times faster than the state-of-the-art single processor vanilla implementation. Its GPU implementation of the mapper graph computation provides an additional 2-fold acceleration in comparison to its CPU counterpart. We have shown in Table 1 that *Mapper Interactive* provides a large number of unique features compared with the state-of-art. We have demonstrated the usefulness of such features via three use cases. We discuss a few possible extensions of *Mapper Interactive* together with challenges and opportunities.

Pushing the scalability boundary. The scalability boundary of mapper graph computation can be pushed even further, especially for larger datasets with more than 10 million points. Assuming sufficient storage and memory, one of the limiting factors is the clustering step during the mapper construction. In this paper, we are able to speed up the clustering process by parallelizing the distance computation on a single (multicore) CPU, as well as a single GPU. One obvious avenue is to distribute the distance computation across multiple GPUs. This task is nontrivial since extensive testing is needed to balance the trade-off between moving data from the CPU to multiple GPUs and merging the results across GPUs.

GPU memory will also become a bottleneck. As the data size increases, the distance matrix grows quadratically. The support for large amounts of memory, on the order of hundreds of gigabytes, is limited on GPUs. We also notice in our implementation, with a large number of intervals, the overhead from moving matrices from CPU to GPU and back increases since this operation is done per interval. When the amount of input data is small, initializing the necessary CUDA kernels is also a severe overhead. As more GPUs are added, this overhead will become more pronounced.

Although DBSCAN is the primary clustering algorithm used in *Mapper Interactive*, an algorithm built for high-performance com-

puting, such as DBSCAN++ [9], could provide additional runtime benefits. Another promising area is using an approximate nearest neighbor library such as *PyNNDescent* [5] instead of computing a the full distance matrix. Finally, datasets of different sizes and dimensionality warrant employing different sets of optimizations. Future work on the tool would entail automatic inference of the optimization strategies based on the input data and system configuration.

Improving extendability for novice and expert users. One of the strengths of *Mapper Interactive* is that it allows users to extend the current analysis and visualization capabilities by adding modules that interface with *scikit-learn*, which also leaves plenty of room for improvement. Many common data analysis libraries follow the well-established API guidelines set forth by *scikit-learn*. Because of this standardization, implementing new libraries is straightforward pragmatically, especially under the expert user setting. Making such extensions accessible for novice users through configuration files is nontrivial and is left for future work.

Parameter selection for the mapper algorithm. Parameter selection for mapper algorithm is a challenging open problem. We use a "best practice" for parameter selection for the mapper algorithm commonly employed by the practitioners. That is, finding a range of parameters where the graph produces stable structures which is enabled by the ability to change parameters on-the-fly in *Mapper Interactive* and immediately visualize and explore the resulting structures. Carriére *et al.* [3] provided theoretical results for automatic parameter tuning. However, their theoretical results require the data to adhere to certain statistical assumptions that are often not applicable to real world datasets. Automatic parameter tuning in practice for the mapper algorithm remains an open problem to be tackled by the TDA community.

Acknowledgments

The authors wish to thank Nathaniel Saul for comments on the initial prototype of our tool; Anantharaman Kalyanaraman, Methun Kamruzzaman, Bala Krishnamoorthy, and Umberto Lupo for valuable suggestions and discussions. This work is partially supported by NSF IIS-1513616 and NSF DBI-1661375.

References

[1] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392:5–22, 2008.

[2] A. Brown, O. Bobrowski, E. Munch, and B. Wang. Probabilistic convergence and stability of random mapper graphs. *Journal of Applied and Computational Topology*, 2020.

[3] M. Carrière, B. Michel, and S. Oudot. Statistical analysis and parameter selection for mapper. *Journal of Machine Learning Research*, 19(12):1–39, 2018.

[4] M. Carriére and S. Oudot. Structure and stability of the one-dimensional mapper. *Foundations of Computational Mathematics*, 18(6):1333–1396, 2018.

[5] W. Dong, C. Moses, and K. Li. Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

[6] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz and Dynagraph — static and dynamic graph drawing tools. In *Graph Drawing Software*, pages 127–148. Springer-Verlag, 2003.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[8] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[9] J. Jang and H. Jiang. DBSCAN++: Towards fast and scalable density clustering. *Proceedings of Machine Learning Research*, 97:3019–3029, 2019.

[10] A. Kalyanaraman, M. Kamruzzaman, and B. Krishnamoorthy. Interesting paths in the mapper complex. *Journal of Computational Geometry*, 10(1), 2019.

[11] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[12] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

[13] P. Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson, and G. Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3:1236, 2013.

[14] D. Müllner and A. Babu. Python Mapper: An open-source toolchain for data exploration, analysis and visualization. http://danifold.net/mapper, 2013.

[15] E. Munch and B. Wang. Convergence between categorical representations of Reeb space and mapper. In S. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[16] C. M. Perou, T. Sørlie, M. B. Eisen, M. Van De Rijn, S. S. Jeffrey, C. A. Rees, J. R. Pollack, D. T. Ross, H. Johnsen, L. A. Akslen, et al. Molecular portraits of human breast tumours. *Nature*, 406(6797):747–752, 2000.

[17] M. Phillips, S. Levy, and T. Munzner. Geomview: an interactive geometry viewer. http://www.geomview.org/, 1993.

[18] A. Rathore, N. Chalapathi, S. Palande, and B. Wang. Topoact: Exploring the shape of activations in deep learning. *Computer Graphics Forum*, 2019.

[19] G. Singh, F. Mémoli, and G. Carlsson. Topological methods for the analysis of high dimensional data sets and 3D object recognition. In *Eurographics Symposium on Point-Based Graphics*, pages 91–100, 2007.

[20] T. Sørlie, C. M. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. B. Eisen, M. Van De Rijn, S. S. Jeffrey, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences*, 98(19):10869–10874, 2001.

[21] 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. *arXiv: 2004.02551*, 2020.

[22] A. E. Teschendorff, A. Miremadi, S. E. Pinder, I. O. Ellis, and C. Caldas. An immune response gene expression module identifies a good prognosis subtype in estrogen receptor negative breast cancer. *Genome biology*, 8(8):R157, 2007.

[23] The GUDHI Project. GUDHI User and Reference Manual. https://gudhi.inria.fr/doc/3.3.0/, 2020.

[24] H. J. van Veen and N. Saul. KeplerMapper. http://doi.org/10.5281/zenodo.1054444, Jan. 2019.

[25] L. J. Van't Veer, H. Dai, M. J. Van De Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. Van Der Kooy, M. J. Marton, A. T. Witteveen, et al. Gene expression profiling predicts clinical outcome of breast cancer. *nature*, 415(6871):530–536, 2002.

[26] Y. Wang, J. G. Klijn, Y. Zhang, A. M. Sieuwerts, M. P. Look, F. Yang, D. Talantov, M. Timmermans, M. E. Meijer-van Gelder, J. Yu, et al. Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. *The Lancet*, 365(9460):671–679, 2005.

Dataset	MI	GT	KM	GT*	KM*	MI/KM	GT*/GT	KM*/KM
ImageNet	3.64	1.82	2.35	6.67	2.62	1.55×	3.66×	1.11×
Cifar	1.23	0.57	0.74	1.90	1.16	1.66×	3.33×	1.57×
Random	0.51	0.34	0.37	1.35	0.71	1.38×	3.97×	1.92×

Table 2: Peak memory usage (in Gigabytes) on three datasets, each with 100K points.

# Pts	**Int**	**MI**	**GT**	**KM**	**GT***	**KM***	**KM/MI**	**GT/GT***	**KM/KM***
ImageNet dataset									
1×10^2	5	0.01	0.27	0.21	0.1	0.21	21.0 ×	2.7 ×	1.0 ×
1×10^3	10	0.05	0.79	0.65	0.11	0.65	13.0 ×	7.18 ×	1.0 ×
1×10^4	20	0.99	4.83	5.52	0.69	2.42	5.58 ×	7.0 ×	2.28 ×
1×10^5	100	22.08	135.29	143.04	13.82	22.26	6.48 ×	9.79 ×	6.43 ×
3×10^5	200	99.74	562.19	578.84	60.99	88.03	5.8 ×	9.22 ×	6.58 ×
Cifar dataset									
1×10^2	5	0.01	0.18	0.31	0.1	0.31	31.0 ×	1.8 ×	1.0 ×
1×10^3	10	0.05	0.45	0.94	0.11	0.96	18.8 ×	4.09 ×	0.98 ×
1×10^4	20	0.82	3.0	3.78	0.46	2.07	4.61 ×	6.52 ×	1.83 ×
1×10^5	100	12.73	43.9	57.51	7.45	15.71	4.52 ×	5.89 ×	3.66 ×
1×10^6	500	265.86	932.77	1182.85	171.97	214.24	4.45 ×	5.42 ×	5.52 ×
3×10^6	1500	931.73	3392.51	3740.08	1025.79	802.77	4.01 ×	3.31 ×	4.66 ×
Random dataset									
1×10^2	5	0.01	0.1	0.51	0.1	0.52	51.0 ×	1.0 ×	0.98 ×
1×10^3	10	0.02	0.43	0.73	0.12	0.74	36.5 ×	3.58 ×	0.99 ×
1×10^4	20	0.62	1.64	2.47	0.5	2.15	3.98 ×	3.28 ×	1.15 ×
1×10^5	100	9.42	22.17	32.6	5.56	13.25	3.46 ×	3.99 ×	2.46 ×
1×10^6	500	185.2	289.11	389.7	113.16	154.86	2.10 ×	2.55 ×	2.52 ×
1×10^7	10000	1738.57	OOM	4556.77	OOM	1963.23	2.62 ×	OOM	2.32 ×

Table 3: Runtime comparison (in seconds) of our implementation vs *KeplerMapper* (KM) and *giotto-tda* (GT) on the *ImageNet*, *Cifar*, and *Random* datasets, respectively. **Int**: intervals. **OOM**: out of memory. **N/A**: not available. **KM/MI**, **GT/GT***, **KM/KM***: speed up factors.

Data Size	**Intervals**	**CPU Version**	**GPU Version**	**CPU/GPU**
ImageNet dataset				
1×10^2	5	0.12	0.05	2.40×
1×10^3	10	0.67	0.06	11.17×
1×10^4	20	2.54	0.80	3.18×
1×10^5	100	22.32	12.93	1.73×
3×10^5	500	69.88	36.22	1.93×
Cifar dataset				
1×10^2	5	0.042	3.95	0.01×
1×10^3	10	0.59	0.08	7.38×
1×10^4	20	2.17	0.53	4.09×
1×10^5	100	20.20	9.18	2.20×
1×10^6	500	331.58	202.87	1.63×
3×10^6	1500	1142.82	753.37	1.52×
Random dataset (128-dimension)				
1×10^2	5	0.05	3.97	0.01×
1×10^3	10	0.71	0.03	23.67×
1×10^4	20	2.21	0.36	6.14×
1×10^5	100	15.07	5.09	2.96×
1×10^6	500	256.36	122.64	2.09×
1×10^7	1500	5234.30	4269.83	1.23×

Table 4: Runtime comparison (in seconds) of our implementation on CPU vs GPU using three testing datasets. **CPU/GPU**: speed up factors.