GhostUMAP2: Measuring and Analyzing (r,d)-Stability of UMAP

Myeongwon Jung, Takanori Fujiwara, and Jaemin Jo

Abstract—Despite the widespread use of Uniform Manifold Approximation and Projection (UMAP), the impact of its stochastic optimization process on the results remains underexplored. We observed that it often produces unstable results where the projections of data points are determined mostly by chance rather than reflecting neighboring structures. To address this limitation, we introduce (r,d)-stability to UMAP: a framework that analyzes the stochastic positioning of data points in the projection space. To assess how stochastic elements—specifically, initial projection positions and negative sampling—impact UMAP results, we introduce "*ghosts*", or duplicates of data points representing potential positional variations due to stochasticity. We define a data point's projection as (r,d)-stable if its ghosts perturbed within a circle of radius r in the initial projection remain confined within a circle of radius d for their final positions. To efficiently compute the ghost projections, we develop an adaptive dropping scheme that reduces a runtime up to 60% compared to an unoptimized baseline while maintaining approximately 90% of unstable points. We also present a visualization tool that supports the interactive exploration of the (r,d)-stability of data points. Finally, we demonstrate the effectiveness of our framework by examining the stability of projections of real-world datasets and present usage guidelines for the effective use of our framework.

Index Terms—Dimensionality reduction, manifold learning, stochastic optimization, reliability, visualization, WebGPU

1 Introduction

We introduce (r,d)-stability to Uniform Manifold Approximation and Projection (UMAP), a framework designed to evaluate the stability of UMAP projections. UMAP is among the most popular dimensionality reduction (DR) techniques, distinguished by its computational efficiency. To achieve efficiency, the official implementation [22] incorporates various acceleration techniques, such as negative sampling [23, 31, 32]. However, these techniques introduce stochasticity, which adds variability to the results. This variability can obscure the distinction between a data point's projection influenced by stochasticity and one that is stable, thereby complicating robust interpretation.

We identified three sources of stochasticity in the UMAP implementation. First, UMAP generates an initial embedding using a stochastic process, such as Spectral Embedding [2], and adds small random noise to the embedding to avoid local minima for optimization. Second, UMAP uses negative sampling where small random samples are drawn at each iteration to simulate repulsive forces among points, instead of using all pairs of points in the dataset. Third, UMAP exploits parallelization without synchronization, facilitated by the Python Numba library [16], which can result in race conditions.

In this work, we define (r,d)-stability as the stability of a point's projection with respect to the variability arising from the first two sources. To assess this, we place hypothetical points, referred to as *ghosts*, around a target point in the initial embedding and observe how their projections evolve through UMAP's optimization process. Due to slight differences in their initial positions and the stochasticity of negative sampling, these projections may diverge from one another. If the ghost projections converge near the target point's projection, we consider the projection to be stable. Otherwise, if they fail to converge, it may indicate that the target point is vulnerable to variability, signaling instability in the final projection. It is worth noting that the instability we discuss here differs from distortion, which inherently arises when reducing high-dimensional data to lower-dimensional embeddings. Rather, our focus is specifically on instability driven by the stochastic algorithms, aiming to identify points sensitive to this randomness.

- *Myeongwon Jung and Jaemin Jo are with Sungkyunkwan University. E-mail: {mw.jung@skku.edu, jmjo@skku.edu}. Jaemin Jo is the corresponding author.*
- *Takanori Fujiwara is with Linköping University. E-mail: tfujiwara@ucdavis.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

We present GhostUMAP2, an efficient DR algorithm to evaluate the (r,d)-stability of points projected by UMAP. To define ghosts and compute their projections efficiently, we sample points from a circle centered at the target point. These ghosts share the identical high-dimensional representation with the target but have perturbed initial positions. We asymmetrically simulate the forces between the original points and ghosts; the ghosts are invisible to the original points. Therefore, the projection of the original points is not affected by our technique. In contrast, we optimize the ghosts with different negative samples, allowing them to diverge from the original points. Finally, the distance to the farthest ghost projection after optimization is used as a measure of the target point's stability.

This work extends a short paper [14] presented at IEEE VIS 2024 with three significant advancements over the previous contribution. First, we provide a formal definition of (r,d)-stability and apply it to UMAP. While doing so, we expand our analysis by accounting for the variability introduced from the random initial embedding in addition to that from negative sampling; indeed, our previous work can be seen as a special case of GhostUMAP2 with $r = 0$. Second, we introduce an adaptive dropping scheme to accelerate the computation of (r,d)-stability, and achieve approximately a 30% speedup over our previous successive halving scheme and up to 60% speedup compared to an unoptimized baseline. Finally, we present an interactive visualization tool, GhostExplorer, for analyzing (r,d)-stability of a projection and guidelines on hyperparameter settings and stability interpretation.

2 Related Work

2.1 Uniform Manifold Approximation and Projection (UMAP)

To explain the details of GhostUMAP2, we provide a concise introduction to the core components of UMAP from a computational perspective. The UMAP algorithm consists of two main phases: graph construction and layout optimization.

Graph Construction. The graph construction phase aims to construct a topological representation of high-dimensional data by building a weighted k-Nearest Neighbor (kNN) graph which combines local fuzzy simplicial sets. Formally, let $X = \{x_1, \ldots, x_N\}$ be the given high-dimensional dataset, with a distance metric $d : X \times X \to \mathbb{R}_{\geq 0}$. For each data point x_i, the set of k-nearest neighbors, $\mathcal{N}_i$, is determined regarding d. Then the weight of a directed edge from x_i to x_j, representing the probability of the edge's existence, is computed as:

$$v_{j|i} = \exp(-\max(0, d(x_i, x_j) - \rho_i)/\sigma_i). \quad (1)$$

Here ρ_i and σ_i are local connectivity parameters, ensuring the assumption that data points are uniformly distributed over a manifold in high-dimensional space. The weight of an undirected edge between x_i and

x_j, denoted v_{ij}, is then computed as the combined probability of at least one directed edge existing: $v_{ij} = (v_{j|i} + v_{i|j}) - v_{j|i} \cdot v_{i|j}$.

Layout Optimization. The layout optimization phase maps the topological graph representation onto a low-dimensional space while preserving its structural properties. This is achieved using a force-directed graph layout algorithm that applies attractive and repulsive forces between points in the low-dimensional space. These forces are derived from gradients of the edge-wise cross-entropy loss, which measures the difference between the high-dimensional and low-dimensional weighted graphs.

The low-dimensional projection $Y = \{y_1, \cdots, y_N\}$ is initialized using spectral embedding. The weight of an edge between points y_i and y_j is defined as:

$$w_{ij} = (1 + a||y_i - y_j||_2^{2b})^{-1}, \tag{2}$$

where a and b are user-specified positive parameters.

The cross-entropy loss, minimized during optimization, is given by:

$$CE = \sum_{i \neq j} \left[v_{ij} \cdot \log\left(\frac{v_{ij}}{w_{ij}}\right) - (1 - v_{ij}) \cdot \log\left(\frac{(1 - v_{ij})}{(1 - w_{ij})}\right) \right]. \tag{3}$$

From this loss function, the attractive and repulsive forces acting on y_i due to y_j are computed as:

$$F_{att}(y_i, y_j, v_{ij}) = \frac{-2abd_{ij}^{2(b-1)}}{1 + d_{ij}^2} v_{ij}(y_i - y_j) \tag{4}$$

$$F_{rep}(y_i, y_j, v_{ij}) = \frac{2b}{(\varepsilon + d_{ij}^2)(1 + ad_{ij}^{2b})} (1 - v_{ij})(y_i - y_j) \tag{5}$$

where $d_{ij} = \|y_i - y_j\|_2$ and ε is a small constant to prevent division by zero. The optimization process iteratively applies these forces to refine the projection. The attractive force is applied between pairs of points connected by an edge in the topological graph, pulling them closer together. In contrast, the repulsive force theoretically needs to be computed between each point and all other points, which is computationally expensive.

To reduce this complexity, UMAP employs negative sampling, where for each point i, a subset of *n_negative_samples* points (*n_negative_samples* $= 5$ by default) is randomly sampled. The repulsive force is applied only between i and the sampled points, substantially improving computational efficiency. However, this stochastic approach introduces randomness to the optimization process, contributing to the inherent variability in UMAP's results.

2.2 Distortion and Instability of DR

To the best of our knowledge, this work is the first to measure and analyze the instability caused by the stochastic nature of DR techniques. In contrast, prior research has primarily focused on the distortions introduced by DR techniques when mapping high-dimensional data to low-dimensional spaces. These distortions have been studied extensively at various levels, including local [6, 18, 33], cluster [10, 26, 29], and global levels [7, 15], resulting in measures that quantify the amount of the distortion. Such measures have been used to assess the quality of DR results and to compare different techniques [5, 9, 24]. Visual interfaces have also been developed to facilitate the analysis and interpretation of the distortions [19, 30].

However, it is important to note that these studies on distortion focus on the relationship between a DR result and the original high-dimensional space, without accounting for the variability introduced during the generation of the result by the DR technique itself. The most closely related work to ours is by Jung et al. [13], where the authors compared multiple t-SNE projections to identify stable clusters across multiple runs. In contrast, our approach measures the instability of a projection in a single run, which is computationally effective.

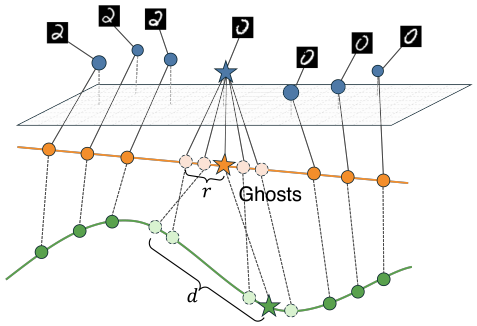

Fig. 1: A motivating example illustrating the (r,d)-stability framework. The figure shows the original high-dimensional data points (colored blue), the initial 1D projection generated by PCA (orange), and the final 1D projection (green). The starred point is vulnerable to stochasticity as its image is ambiguous between two classes (digits zero and two). We attach ghosts to its initial projection (orange dashed line) and jointly optimize the original and ghost projections. As a result, the distribution of ghost projections can highlight the stochastic instability of the point.

3 GhostUMAP2

We introduce GhostUMAP2, incorporating the concept of (r,d)-stability into UMAP. With a motivating example, we define ghost projections to measure the (r,d)-stability and elaborate on the layout optimization process for ghost projections. We further present an adaptive dropping scheme to reduce the computational overhead introduced by the ghost projections. We then present a visualization tool, GhostExplorer, to analyze the results of GhostUMAP2.

3.1 A Motivating Example

The layout optimization process of UMAP starts with an initial projection created by simpler DR methods, such as Principal Component Analysis (PCA) [8] or Spectral Embedding [16]. Fig. 1 illustrates the relationships between the original data (colored blue), initial projection (orange), and final projection (green). For illustration, consider a scenario where a 1D manifold in 3D space, conceptualized as a hypothetical curved line formed by the blue points in Fig. 1, is projected onto a 1D space. In this example, each original high-dimensional data point corresponds to a hand-written digit 0 or 2.

During the optimization process, the initial projection (orange line in Fig. 1) is iteratively annealed to the final projection (the green line) to best describe the manifold in the high-dimensional space where the goodness of fit is computed as UMAP's objective function (Eq. (3)). However, since the optimization process is stochastic, the final projection is subject to variability. For example, the starred point in Fig. 1 is pushed more toward the cluster of digit 0 by chance, misleading the user into concluding the starred point is clearly different from digit 2. Our framework would identify that this starred point is not (r,d)-stable.

To specify data points prone to variability, we introduce clones of a target point, called ghosts, which are shown as dotted orange circles in Fig. 1. In the initial projection, we randomly place ghosts around the target point within a distance r where r is a user-specified hyperparameter. In practice, UMAP is often used to generate a 2D projection and we place ghosts within a r-radius circle centered at the target point. While these ghosts have different projection positions, they still have the same high-dimensional vector as the target point. The ghosts can be viewed as alternative projections of the target point with different starting positions created by adding perturbations. We then jointly optimize the projection positions of the ghosts as well as the target point (detailed in Sec. 3.3). The result is shown as dotted green circles in Fig. 1. We can see that two ghosts are projected closer to the cluster of digit 2 and the other two closer to digit 0, implying the original projection (the green star) is highly subject to variability, and thus, its position should be interpreted with care.

3.2 (r,d)-Stability of a Point in UMAP

We formulate the (r,d)-stability of a point in a UMAP projection. Let $X = \{x_i \mid i \in [1, ..., N]\}$ be a set of high-dimensional input vec-

tors where $x_i \in \mathbb{R}^D$ (D: the number of dimensions) and $Y = \{y_i \mid i \in [1,...,N]\}$ be a set of initial projections of X where $y_i \in \mathbb{R}^2$. The optimization process of UMAP can be seen as a function that maps the initial projection Y to the final projection Y', i.e., $Opt : (Y, \mathscr{W}, \mathscr{H}) \mapsto Y'$ where $\mathscr{W}$ is the weighted kNN graph of X (refer to Sec. 2.1) and $\mathscr{H}$ is a set of UMAP's hyperparameters. We assume both Y and Y' are normalized into a range $[[0,1],[0,1]]$

In GhostUMAP2, each point x_i has M ghost projections, $g_{i_k} \in \mathbb{R}^2$ ($k \in [1,...,M]$). Each ghost's projection is initialized to a random position in a circle of radius r around the initial projection of the target y_i:

$$g_{i_k} \leftarrow SampleCircle(y_i, r) \tag{6}$$

We chose a circle over other shapes for simplicity. While one could sample ghost projections exactly at a distance r from the target, i.e., from the circumference of a circle, we opted to sample from within the circle's interior for two reasons. First, our intention was to approximate a ball rather than a shell. Second, this choice aligns with UMAP's initialization, where each point's projection is given a small random perturbation sampled from within a bounded range.

Let $G = \{g_{i_k} \mid i \in [1,...,N],\ k \in [1,\ldots,M]\}$ be the set of the ghost projections of all data points in X. The optimization of GhostUMAP2 is a function that jointly optimizes the initial projections of original points ($y_i \in Y$) and ghosts ($g_{i_k} \in G$):

$$LayoutOptimization : (Y, G, \mathscr{W}, \mathscr{H}) \mapsto (Y', G') \tag{7}$$

where G' is the final projection of the ghosts. We let d_i denote the maximum Euclidean distance between the final projection positions of the target point, y'_i, and its ghost, g'_{i_k}:

$$d_i := \max_k \|y'_i - g'_{i_k}\|_2 \tag{8}$$

We define x_i (r,d)-stable if $d_i \le d$ with a user-defined threshold d.

The hyperparameter r can be seen as "stability loads" given to each point. Increasing r will make fewer points be (r,d)-stable for the same d. In contrast, d can be adjusted after the optimization without recomputation and determines how generous we are about the instability. Decreasing d will make fewer points be (r,d)-stable. In practice, the user may want to decrease d as much as possible while keeping a sufficient number of points remaining (r,d)-stable. Then, the user can refer to these stable points to interpret the projection results or the unstable points to investigate the cause of instability (refer to our use cases in Sec. 6). Both r and d are in the range $[0,1]$ as they are defined in normalized coordinates.

Suppose a perfectly deterministic DR algorithm that projects x_i to a fixed point y'_i regardless of its initial projection y_i without any negative sampling. In this ideal case, x_i is $(1,0)$-stable as there is no variability in y'_i even though we impose an infinite amount of perturbation on y_i. On the other extreme, suppose a fully random algorithm that maps x_i to a random point. In this worst possible case, x_i is $(0,1)$-stable as there is no bound in the projection y'_i even though we impose no perturbation.

3.3 Ghost Generation and Joint Optimization

The optimization process in GhostUMAP2 consists of three phases: ghost generation, layout optimization, and adaptive ghost dropping (Alg. 1). This subsection focuses on the first two phases, ghost generation and layout optimization. In the ghost generation phase, we randomly generate M ghosts by following Eq. (6).

We can attach the ghosts not to the very initial projection but after running a few layout optimization iterations of UMAP, which we call lazy generation. Such deferred generation can have three benefits: First, during the early epochs of layout optimization, the UMAP's learning rate α is at its peak, resulting in rapid changes to the projection. These early decisions often dominate the process, making it challenging to evaluate the influence of randomness in later stages. Second, the initial projection is often much different from the final projection, and thus, measuring the stability using the initial projection may be less meaningful. After a few optimization iterations, the projection better fits the manifold we want to approximate, which can be a more meaningful starting point. Third, delaying ghost generation reduces computational overhead as we can start optimization in a later epoch, thereby reducing the runtime. In our implementation, this is controlled by a hyperparameter, $lazy\_gen \in [0,1)$, which is multiplied to n_epochs to determine at which epoch we generate ghosts.

We set three requirements for layout optimization. First, the optimization of ghost projections must not interfere with the original projections. This ensures that original points are projected as if the ghosts were absent, thereby producing projections identical to those of the original UMAP. Second, ghost projections must accurately represent their corresponding projections of the original points. To achieve this, when generating ghosts, we add the perturbation only to the projection positions, and all ghosts still have the same high-dimensional vector, x_i, as their target point. Third, separate negative samples should be used for the original and ghost projections to evaluate the impact of negative sampling on the optimization process.

The joint optimization of original and ghost projections is outlined in Lines 10 and 11 in Alg. 1. We first optimize the original projections Y as with the standard UMAP (Alg. 2). Note that, to ensure no influence from the ghost on the original projections, 1) the weighted kNN graph, $\mathscr{W}$, is generated without using ghosts and 2) ghosts are not chosen by negative sampling.

The optimization process for ghost projections is detailed in Alg. 3. We introduce two modifications when compared with Alg. 2. First, the attractive and repulsive forces are computed with and applied to a ghost, g_{i_k}, instead of y_i (Lines 5 and 8 in Alg. 3). This ensures that ghost projections serve as alternatives to the original projections. Second, we negative-sample only the original projections again (i.e., ghosts are not

Algorithm 1 GhostUMAP Layout Optimization

```
1:  function LayoutOptimization(Y, G, W, H)
2:      α ← 1.0      (note: α is a learning rate)
3:      e_gen ← ⌈n_epochs × lazy_generation⌉
4:      e_drop ← ⌈n_epochs × drop_start⌉
5:      G ← None
6:      D_i ← 0 for all i ∈ [1,...,N]
7:      for e ← 1 to n_epochs do
8:          if e ≥ e_gen and G is None then
9:              G ← SampleCircle(Y, r)
10:         OptimizeOriginalLayout(Y, W)
11:         OptimizeGhostLayout(Y, G, W)
12:         D ← UpdateDistances(Y, G, D)
13:         if e ≥ e_drop then
14:             DropGhosts(G, D)
15:         α ← 1.0 − e/n_epochs
16:     return Y, G
```

Algorithm 2 Layout Optimization for Original Projections

```
1: function OptimizeOriginalLayout(Y, W)
2:     for all ([i, j], v_ij) ∈ W do
3:         f_att ← F_att(y_i, y_j, v_ij)
4:         (y_i, y_j) ← (y_i + α · f_att, y_j − α · f_att)
5:         for n ← 1,...,n_negative_samples do
6:             l ← random sample from {1,...,N}\{i}
7:             y_i ← y_i + α · F_rep(y_i, y_l, v_il)
```

Algorithm 3 Layout Optimization for Ghost Projections

```
1: function OptimizeGhostLayout(Y, G, W)
2:     if G is None then return
3:     for all ([i, j], v_ij) ∈ W do
4:         for all k ← 1,...,n_ghosts do
5:             g_ik ← g_ik + α · F_att(g_ik, y_j, v_ij)
6:             for n ← 1,...,n_negative_samples do
7:                 l ← random sample from {1,...,N}\{i}
8:                 g_ik ← g_ik + α · F_rep(g_ik, y_l, v_il)
```

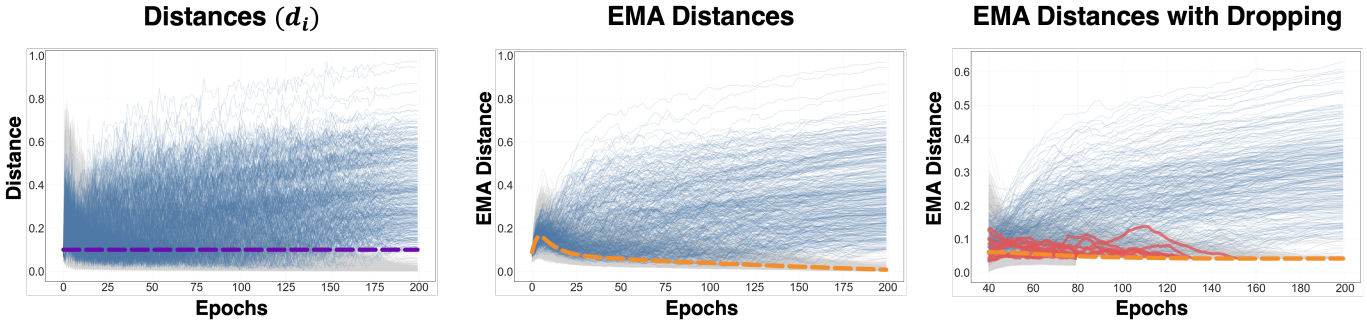

Fig. 2: Distance changes of stable (gray) and unstable (blue) projections by optimization epochs. **Left**: We choose points with $d_i \geq 0.1$ in the final projection as unstable points, whose threshold ($d = 0.1$) is shown as the purple dashed line. **Middle**: We use Exponential Moving Average (EMA) to mitigate the fluctuation and set the dropping threshold as the mean of the averaged distances (orange dashed line). **Right**: In practice, we freeze the D_i of dropped points (orange dashed line). The false positives, i.e., the points that are unstable but dropped mistakenly, are shown as red lines. The dropping operation started at epoch 80.

Algorithm 4 Adaptive Dropping Scheme

```
1:  function UpdateDistances(Y, G, D)
2:      if G is None then return end if
3:      j ← ⌈M × sensitivity⌉
4:      for all i ∈ {id(g_i) | g_i ∈ G} do
5:          d_i ← distance from y_i to the j-th farthest ghost g_{i_k}
6:          D_i ← β · d_i + (1 − β) · D_i
7:      return D
8:
9:  function DropGhosts(G, D)
10:     Compute threshold: τ ← mean of D
11:     Keep unstable points in G: G ← {g_i ∈ G | D_i ≥ τ}
```

used as negative samples), which would be different from the negative samples used for the original point (Lines 6-8 in Alg. 3). This allows each ghost projection to model the outcome that would result if different negative samples were assigned to its corresponding original projection.

The joint optimization of the original and ghost projections can be summarized at a high level as follows: the original projections are optimized in a way that they are unaware of ghosts. Each ghost is also unaware of other ghosts but is optimized as if it were the original projection of a point.

3.4 Adaptive Dropping Scheme

Jointly optimizing the ghost projections with the original projection requires extra time to compute. Roughly speaking, having M ghosts (by default, $M = 16$) is equivalent to optimizing M times more data points than the original ones, slowing down the process significantly. To reduce such performance overhead, in the previous work [14], we introduced a successive halving (hereafter, halving) scheme. The halving scheme sorts data points by the positional variance in the original and ghost projections and performs a halving operation that discards the ghosts of half of the points with the lowest variance (i.e., stable points).

One significant constraint of the halving scheme is that the user should determine epochs at which they want to perform the halving operation, i.e., a halving schedule, before they see how the optimization proceeds. This constraint leads to two drawbacks. First, a fixed schedule may miss the opportunity to drop ghosts of the stable points as early as possible since the dropping needs to wait until the next halving epoch is reached. Second, this scheme only allows for a fixed ratio of data points' ghosts to be dropped, merely guaranteeing a fixed amount of speedups. For example, if we schedule three halving operations, only 87.5% points will be dropped eventually, even though more points can be found to be sufficiently stable.

To address the limitation of the halving scheme, we introduce an adaptive dropping scheme that identifies stable points as early as possible, as outlined in Alg. 4. Rather than dropping a fixed number of ghost projections at a certain epoch, our new scheme constantly checks if a point is stable enough by monitoring d_i—the maximum distance between its original projection and ghosts (Eq. (8)). At the beginning, $d_i \leq r$ because all ghosts are in a r-radius circle centered at the target point. However, as the optimization proceeds, d_i can fluctuate. We are interested in identifying points whose d_i is maintained less than a certain threshold and dropping their ghosts. In contrast, points with large d_i should avoid dropping their ghosts as they are of interest.

To establish a dropping criterion, we examined how d_i of a stable or unstable point changes during optimization, as shown in Fig. 2. In the left chart, the gray and blue lines depict d_i of stable and unstable points, respectively. For illustration, we chose points with $d_i \geq 0.1$ in the final projection as unstable points. We highlighted the threshold as a purple dashed line. Note that although there is a fluctuation, d_i of a stable point (gray) gradually converges, while d_i of an unstable point does not. We leverage this observation of the convergence to identify stable points. Instead of directly referring to d_i, we compute the smoothed distance D_i by applying exponential moving average (EMA) [3] to d_i with a smoothing factor β (0.2 by default), as shown on Line 6 of Alg. 4. This smoothing reduces volatility, as shown in the middle plot of Fig. 2, allowing a clearer separation of stable and unstable points. For each iteration, we use the mean of D_i as a dropping threshold, $\tau = mean(D_i)$, and drop the ghosts of the points whose smoothed distance D_i is less than τ (Lines 10–11 in Alg. 4). τ over iterations is shown as the orange dashed line in the middle chart of Fig. 2. One can consider using medians instead of means. However, doing so is essentially similar to the halving scheme. Furthermore, as medians are more robust than means, we found them to be overly conservative, often retaining even very stable points.

One practical concern on τ is the inflation of the threshold as stable points are dropped. For example, in practice, about half of the points are dropped in the first dropping operation as τ is set to $mean(D_i)$. In the next iteration, if we compute τ again only with the remaining half of the points, i.e., points with higher D_i, τ will rapidly increase. We observed that such an inflation of τ degrades the quality of dropping by mistakenly discarding unstable points (refer to the red lines in Fig. 2-Right). To prevent this issue, instead of eliminating D_i of the points with dropped ghosts, we include their D_i when computing τ while keeping their D_i as the same value even after the ghost dropping.

Another concern is that since d_i is determined by the farthest ghost projection, it is sensitive to outliers. Indeed, we observed that certain negative samples push the ghost projections far away from the original projection by chance. To prevent such influence by outliers, we introduce a hyperparameter $sensitivity \in [0, 1]$ (0.9 by default) that determines the percentile of the ghost projection distances to consider. If it is set to 1.0, we consider the distance to the farthest ghost projection. When set to 0.9, we use the 90th percentile of the distances to ghosts.

The last question is about when to start the dropping operation. One can start dropping operations immediately after ghosts are initialized. This may allow for reducing the runtime by dropping the ghosts as early as possible but at the risk of mistakenly dropping the ghosts of

Code 1: The Python Interface of GhostUMAP2

```
1  from ghostumap import GhostUMAP2
2  from data import load_data
3
4  X, y, legend = load_data()
5  gmap = GhostUMAP2()
6  O, G, survived = gmap.fit_transform(X, n_ghosts=16, r=0.1)
7
8  # N: the number of data points, i.e., X.shape[0]
9  # O: original projections of shape (N, 2)
10 # G: ghost projections of shape (N, n_ghosts, 2)
11 # survived: a bitmask of length N indicating if point i has
       not been dropped.
12
13 unstable = gmap.get_unstable(d=0.1)
14 # unstable: a bitmask of length N indicating if point i is
     unstable with respect to d.
15
16 gmap.visualize(label=y, legend=legend)
```

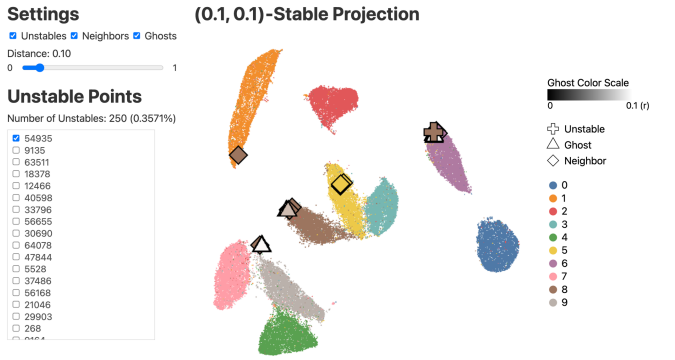

Fig. 3: GhostExplorer, a visual interface for GhostUMAP2, shows original and ghost projections on a zoomable scatterplot using color and symbol encodings. The control panel allows users to adjust the stability threshold d interactively.

unstable points since the projection is premature. To address this, we start the dropping operation after a certain portion of epochs is done, specified by a hyperparameter, $drop\_start \in [0,1]$ (0.4 by default). We will investigate the effect of these hyperparameters in Sec. 4.2.

The GhostUMAP2 implementation is written in Python and distributed as a Python package (`ghostumap`), making it easily installable via the Python Package Index (`pip`) or Anaconda. The package employs an API consistent with UMAP, such as `gmap.fit_transform` in Code 1, with additional hyperparameters for our technique, such as the number of ghosts. To enhance efficiency, we also utilized the Numba package in the same way as UMAP.

3.5 GhostExplorer: A Visualization Tool for GhostUMAP2

To facilitate the interpretation of ghost projections, we also present GhostExplorer, an interactive visualization tool for GhostUMAP2 (Fig. 3), implemented as a Jupyter Notebook widget and accelerated by WebGPU. It can be invoked from a GhostUMAP2 instance, e.g., `gmap.visualize()` (Line 16 in Code 1).

GhostExplorer visualizes the projections of the original points and ghosts on a zoomable scatterplot (Fig. 3), with optional color encoding for data labels (if available). The user can adjust d using a slider in the control panel on the left. Fig. 4 illustrates identified unstable points for different values of d. The control panel also lists unstable data points, i.e., data points whose $d_i > d$.

To inspect individual ghosts of a specific target point, the user can click on a point in the scatterplot or the checkbox of a point in the list. This highlights the target point and its ghost projections in the scatterplot using larger symbols: a cross for the original projection, triangles for the ghost projections, and diamonds for the projections of its neighbors in the high-dimensional space (see Fig. 6). Ghost projections can be further color-encoded based on their distance from the original projection in the initial projection; we used lightness for this optional encoding to avoid conflict with hues often used for encoding class labels. The user can also toggle the visibility of unstable points through the visibility checkboxes at the top of the control panel, leaving only (r,d)-stable points in the scatterplot. In the supplementary material, we present examples of analytic tasks that GhostExplorer supports. While GhostExplorer benefits from WebGPU acceleration, it remains subject to visual scalability issues inherent to scatterplots such as overplotting. Thus, an interesting extension would be to employ more scalable idioms as proposed in prior studies [12, 20, 21].

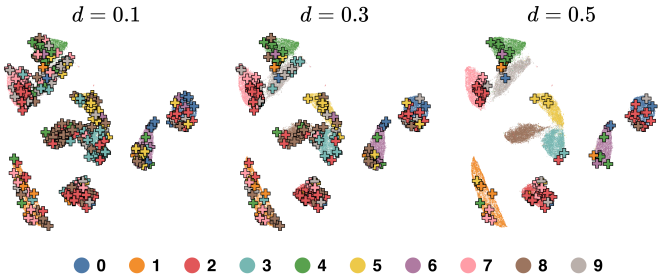

Fig. 4: Points identified as unstable with different stability thresholds d. More points are identified as unstable for a small, thus strict, threshold.

4 Quantitative Experiments

In this section, we perform two quantitative experiments. The experiments had two primary objectives: (1) to evaluate the overhead introduced by GhostUMAP2 and assess how the adaptive dropping scheme mitigates this overhead while maintaining accuracy, and (2) to analyze the impact of hyperparameters on both runtime and accuracy.

4.1 Performance Benchmark

The benchmark study was conducted to assess the effectiveness of the adaptive dropping scheme compared to the halving scheme [14].

Datasets. We used eight datasets, with varying sizes and data types, such as tables, text, and images (Tab. 2), that were widely used in evaluating DR techniques [11, 14, 31].

Methods and Hyperparameters. We compared four methods: UMAP, GhostUMAP2 without any ghost reduction method (hereafter, no reduction), with the successive halving (hereafter, halving), and with the adaptive dropping (hereafter, adaptive). For the UMAP hyperparameters for all methods, we used the default values specified in the official documentation [22]: $n\_neighbors = 15$, $min\_dist = 0.1$, and Spectral Embedding for initialization. For GhostUMAP and GhostUMAP2, the number of ghosts (M) was set to 16. For GhostUMAP with halving, the schedule hyperparameter was set to [50, 100, 150] as proposed in the previous work [14]. For GhostUMAP2 with the adaptive dropping, the hyperparameters were set to the default values: $lazy\_gen = 0.2$, $drop\_start = 0.4$, $\beta = 0.2$, and $sensitivity = 0.9$.

Note that UMAP sets the number of epochs (n_epochs) based on the dataset size: 200 epochs for datasets with over 10,000 data points and 500 epochs for smaller ones. As both $lazy\_gen$ and $drop\_start$ are multiplied to n_epochs, the epoch when the ghosts were generated ($lazy\_gen$) and when dropping started ($drop\_start$) varied depending on the dataset size. For example, when $lazy\_gen = 0.2$, the ghosts were generated at the 100th epoch for the MNIST dataset ($500 \cdot 0.2 = 100$).

Measures. We measured two metrics: the runtime (for all methods) and accuracy (for the halving and adaptive methods). We measured the runtime to inspect the computational overhead the ghost projections imposed. We did not include the runtime to produce the weighted kNN graph as it was common for all methods.

For the halving and adaptive methods, we measured the F1 score to investigate to what extent they maintain unstable points throughout the

Table 1: Benchmark results of different methods across the target datasets. In terms of runtime, our new adaptive dropping method achieved a speed up of 2.4x on average compared to the baseline with no ghost reduction and of 1.4 compared to the halving method. The halving and adaptive methods exhibited similar recall scores. In contrast, the adaptive method precisely identified unstable points, showing significantly better F1 scores than the halving method. For F1 and recall scores, we report the standard deviations in parentheses.

	Completion time in seconds (Speedup by ghost reduction)				F1		Recall	
Dataset	UMAP	GhostUMAP2 (No reduction)	GhostUMAP (Halving)	GhostUMAP2 (Adaptive)	GhostUMAP (Halving)	GhostUMAP2 (Adaptive)	GhostUMAP (Halving)	GhostUMAP2 (Adaptive)
C. elegans	3.19	32.58	20.65 (1.58)	**14.58 (2.23)**	0.08 (±0.01)	**0.89** (±0.05)	0.87 (±0.04)	**0.89** (±0.04)
Optical Recognition	2.90	27.49	17.75 (1.55)	**12.61 (2.18)**	0.05 (±0.01)	**0.95** (±0.06)	**0.94** (±0.06)	0.91 (±0.10)
MNIST	10.18	130.15	78.30 (1.66)	**51.71 (2.52)**	0.10 (±0.00)	**0.94** (±0.01)	0.94 (±0.02)	**0.95** (±0.02)
Fashion-MNIST	10.57	135.26	79.78 (1.70)	**54.29 (2.49)**	0.04 (±0.00)	**0.89** (±0.02)	0.82 (±0.02)	**0.85** (±0.02)
Kuzushiji-MNIST	10.40	134.39	80.30 (1.67)	**54.86 (2.45)**	0.08 (±0.00)	**0.92** (±0.01)	0.91 (±0.02)	**0.92** (±0.01)
20NG	3.73	37.41	23.14 (1.62)	**17.16 (2.18)**	0.08 (±0.01)	**0.88** (±0.03)	0.85 (±0.03)	0.85 (±0.03)
AG News	17.30	223.99	132.09 (1.70)	**89.26 (2.51)**	0.09 (±0.01)	**0.88** (±0.02)	0.88 (±0.02)	**0.90** (±0.01)
Amazon Polarity	57.79	795.45	452.61 (1.76)	**325.24 (2.45)**	0.09 (±0.00)	**0.87** (±0.01)	**0.84** (±0.01)	0.82 (±0.01)

Table 2: Summary of datasets used in the benchmark study.

Dataset	Size ($N\times$dims)	Type
C. elegans [25]	6,188×50	Table
Optical Recognition [1]	5,620×64	Image
MNIST [17]	70,000×784	Image
Fashion-MNIST [34]	70,000×784	Image
Kuzushiji-MNIST [4]	70,000×784	Image
20NG [27]	18,846×768	Text
AG News [35]	120,000×768	Text
Amazon Polarity [35]	400,000×768	Text

Table 3: Effect of the number of ghosts (M)

	C. elegans		MNIST		AG News	
M	Time (s)	F1	Time	F1	Time	F1
0	3.19	-	10.18	-	17.30	-
8	9.28	0.87	32.04	0.91	52.42	0.84
<u>16</u>	13.93	**0.89**	50.67	**0.94**	87.43	**0.88**
32	23.58	0.85	89.44	0.91	153.89	**0.88**
64	44.52	**0.89**	163.76	**0.94**	299.96	0.86
128	165.00	0.84	322.84	**0.94**	1185.41	0.87

halving or adaptive dropping operations, compared to the case without these reduction operations. To compute the F1 score, we need to define the ground truth and predicted sets of unstable points. We defined the ground truth set as the points that did not satisfy (r,d)-stability (i.e., unstable) when no ghost reduction was applied.

For the adaptive method, we set the predicted set of unstable points as the points that have ghsots, i.e., survived after the adaptive dropping operations, and whose $d_i > d$. Throughout the benchmark, we used $r = 0.1$ for ghost perturbation and $d = 0.1$ as the threshold for unstable points. In contrast, the halving method does not explicitly produce a predicted set of unstable points [14]. Instead, it simply reduces the number of ghosts by dropping the most stable points at a given time point, leaving a set that is expected to contain all unstable points. Therefore, for the predicted set, we used all points that survived after the three halving operations ($\frac{1}{8}$ of all points). As a result, the predicted set for the halving method was much larger than the ground truth set (usually less than 1% of points), resulting in poor F1 scores due to low precision scores. To mitigate such bias, we also report the recall scores of halving and adaptive methods.

Setting. We used a desktop equipped with an AMD Ryzen 9 7900 CPU, with 3.7 GHz base clock, 12 cores, 24 threads, and 64GB of DDR5 RAM. GPUs were not used in the benchmark.

Result and Discussion. The results of the benchmark are summarized in Tab. 1. For runtime, GhostUMAP2 without ghost reduction introduced significant computational overhead, increasing the runtime by approximately 11.9 times on average compared to the conventional UMAP, which was closely aligned with the number of ghosts generated (i.e., $M = 16$). The halving method resulted in a runtime 7.2 times longer than the conventional UMAP on average, representing a 1.7 times speedup over GhostUMAP2 without ghost reduction. The adaptive method further improved efficiency, with an average runtime 5.0 times longer than the conventional UMAP, achieving 2.4 and 1.4 times speedups over no ghost reduction and the halving method, respectively.

Both the halving and adaptive methods achieved high recall scores: 0.88 and 0.89 on average, respectively. The high recall score of the halving method is expected as it generated a large predicted set of unstable points. Although the adaptive method generated a much smaller predicted set, it achieved similar recall scores to the halving method. However, these two methods showed significantly different F1 scores. The adaptive method achieved an average F1 score of 0.90, while the halving method scored 0.08, illustrating high scores for both precision and recall of the adaptive method.

Overall, the results suggest that the adaptive method not only provides substantial speedup (2.4 times on average) from the cases with no ghost reduction but also delivers a high F1 score, underscoring its efficacy in evaluating the (r,d)-stability of data points.

4.2 Effect of Hyperparameters

GhostUMAP2 introduces five hyperparameters: the number of ghosts per point ($M = 16$ by default), when to spawn ghosts ($lazy\_gen = 0.2$ by default), when to start dropping ($drop\_start = 0.4$ by default), the smoothing factor in the exponential moving average ($\beta = 0.2$ by default), and the sensitivity in dropping ($sensitivity = 0.9$ by default). We set the default values of these hyperparameters through extensive grid searches on multiple datasets.

Dataset and Setting. We report the results on three datasets from the previous benchmark, *C. elegans*, MNIST, and AG News, with various sizes and types; additional results on different datasets can be found in the supplementary material. We used the same measures and settings as the previous benchmark. That is, the ground truth for measuring F1 score was set as the points whose $d_i > 0.1$ when no ghost reduction was applied. We systematically evaluated the impact of each hyperparameter while keeping the other hyperparameters with default values. The specific values tested for each hyperparameter were as follows (default values are underlined):

- $M \in \{8, \underline{16}, 32, 64, 128\}$
- $lazy\_gen \in \{0.1, 0.15, \underline{0.2}, 0.25, 0.3\}$
- $drop\_start \in \{0.3, \underline{0.4}, 0.5, 0.6, 0.7\}$
- $\beta \in \{0.01, 0.05, 0.1, \underline{0.2}, 0.3, 0.4\}$
- $sensitivity \in \{0.8, \underline{0.9}, 1.0\}$

Result and Discussion. Tabs. 3 to 7 present the benchmark results, with the default values underlined for clarity. For the number of ghosts

Table 4: Effect of *lazy_gen*

	C. elegans		MNIST		AG News	
lazy_gen	Time (s)	F1	Time	F1	Time	F1
0.10	18.05	**0.89**	64.93	**0.94**	113.17	**0.88**
0.15	15.88	0.76	58.35	0.93	101.79	0.86
0.20	13.93	**0.89**	50.67	**0.94**	87.43	**0.88**
0.25	12.42	0.88	42.19	0.93	73.05	0.86
0.30	10.68	0.88	35.15	0.89	60.44	0.86

Table 5: Effect of *drop_start*

	C. elegans		MNIST		AG News	
drop_start	Time (s)	F1	Time	F1	Time	F1
0.3	10.80	0.85	35.75	0.91	72.45	0.86
0.4	13.93	**0.89**	50.67	0.94	87.43	**0.88**
0.5	17.79	**0.89**	65.93	**0.95**	112.61	0.87
0.6	20.87	0.84	79.94	0.92	142.44	0.86
0.7	24.86	0.83	96.27	0.89	163.54	0.79

Table 6: Effect of β

	C. elegans		MNIST		AG News	
β	Time (s)	F1	Time	F1	Time	F1
0.05	15.49	0.88	55.10	**0.95**	94.42	**0.88**
0.10	14.62	0.85	52.44	0.93	91.70	0.87
0.20	13.93	**0.89**	50.67	0.94	87.43	**0.88**
0.30	13.90	**0.89**	49.17	0.94	84.40	0.83
0.40	13.36	0.81	48.67	0.91	84.15	0.85

Table 7: Effect of *sensitivity*

	C. elegans		MNIST		AG News	
sensitivity	Time (s)	F1	Time	F1	Time	F1
0.8	13.37	0.86	49.79	0.90	85.61	0.84
0.9	13.93	**0.89**	50.67	**0.94**	87.43	**0.88**
1.0	14.49	**0.89**	51.80	0.92	89.90	**0.88**

M (Tab. 3), we observed that the runtime increased with the value of M almost proportionally. This is expected since optimizing M ghosts per data point slows down each optimization iteration by approximately $(M+1)$ times. In contrast, no clear trend was observed regarding the F1 score. However, using too few ghosts may limit the exploration of alternative projections using GhostExplorer, potentially compromising their usefulness. Consequently, we propose $M = 16$ as the default value, as it strikes a reasonable balance between runtime and usefulness.

For *lazy_gen* (Tab. 4) and *drop_start* (Tab. 5), we found their impacts on the runtime. The larger value for *lazy_gen* defers ghost generation to later epochs, reducing the runtime. In contrast, the larger value for *drop_start* delays the adaptive dropping, increasing the runtime. Large values for these parameters showed a subtle but negative impact on F1 scores. Based on these observations, we recommend setting the default values of *lazy_gen* and *drop_start* to 0.2 and 0.4, respectively.

For the smoothing factor β (Tab. 6), despite its name, a larger value reduces the amount of smoothing [3], causing the moving average of distances to fluctuate more. This increased fluctuation in the average distance can lead to a higher rate of mistakenly dropped points, i.e., false positives. In fact, we observed that an excessively larger value of β (e.g., 0.4) reduced runtime but produced a worse F1 score. Based on these observations, we recommend a smoothing factor of 0.2 as it efficiently drops ghosts while limiting false positives.

Finally, for *sensitivity* (Tab. 7), a value of 0.8 resulted in the fastest runtime but decreased the F1 score by approximately 0.04. This decrease of F1 score is due to an excessive amount of data points that are considered outliers. On the other hand, when *sensitivity* is 1.0, we also see the decrease of F1 score for the MNIST dataset. We can expect that, for this dataset, the computation of d_i was heavily influenced by outliers. We recommend a value of 0.9 for *sensitivity*, as it maintains a high F1 score by avoiding the influence from outliers in general.

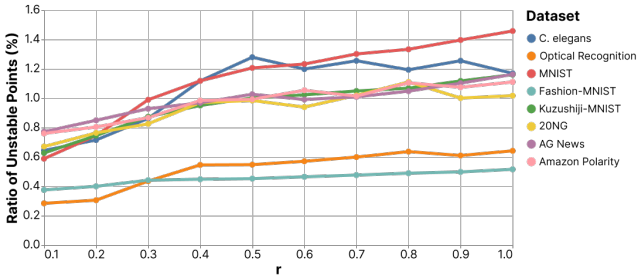

Fig. 5: Ratio of unstable points across different values of r for eight datasets in the Tab. 2. As r increases, the proportion of points classified as unstable (i.e., $d_i \geq 0.1$) also increases, indicating that larger perturbations tend to amplify projection variability.

5 Usage Guidelines

This section provides usage guidelines for our framework, informed by the lessons learned from developing and experimenting with it.

5.1 Considerations on the Choice of r and d

The hyperparameter r simulates variability given in the initial projection. As shown in Fig. 5, a larger r results in more data points being classified as unstable when d is fixed ($d = 0.1$). Another important finding regarding r is that the distance between an unstable point and its ghosts in the final projection is positively correlated with r, but the correlation is weak—with a Pearson correlation coefficient of up to approximately 0.15. This suggests that the hypothetical ball we assumed around the initial projection undergoes complex transformations during optimization, leading to non-uniform distortions that weaken the spatial relationship between initial and final positions.

The value of r should be selected based on the intended use of GhostUMAP2. If the goal is to identify data points that are sensitive to stochasticity, a small value such as 0.1 (the default) is typically sufficient. In contrast, if the focus is on isolating only the most robust points, a larger value, such as 1.0, can be used, allowing one to remove all points identified as unstable and leave the most stable points.

In contrast to r, the hyperparameter d can be tuned without recomputation of the projection, which means that the user can interactively choose its value. Increasing its value leads to a higher number of unstable points. If the goal is to remove unstable points from the projection, one can gradually increase d from zero until a sufficient number of points remain for meaningful interpretation, such as identifying clusters. Empirically, we observe that in most datasets, over 90% of points have $d_i < 0.01$, suggesting that setting $d \approx 0.01$ retains the majority of the data. Alternatively, if the aim is to highlight unstable points as potentially meaningful outliers, which are useful for exploring subtle cluster relationships, a higher threshold such as $d = 0.1$ may be used, which typically classifies about 1% of points as unstable.

5.2 Considerations on Other Hyperparameters

For the hyperparameters of adaptive dropping, we generally recommend using the default values identified through our experiments. If the goal is to more thoroughly explore stochastic variation, M, the number of ghosts (default: 16), can be increased to values, such as 32 or 64. To accelerate computation while preserving the ability to detect instability, we recommend increasing *lazy_gen* (default: 0.2) or decreasing *drop_start* (default: 0.4), rather than reducing M from its default. We found that setting either parameter to 0.3 maintains the accuracy of unstable point detection while achieving a meaningful speedup of approximately 1.3×. We generally do not recommend increasing β or decreasing sensitivity to accelerate computation, as the resulting degradation in accuracy outweighs the speedup.

5.3 Stability Assessment Using Ghosts

We present guidelines on assessing the stability of a data point by interpreting the distribution of its original projection and ghosts. We

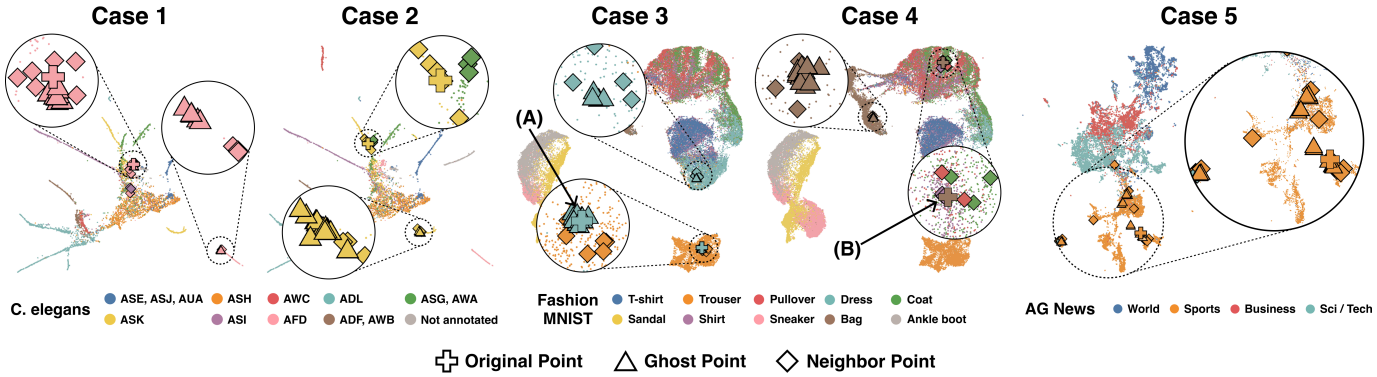

Fig. 6: Projections of the original (cross), their ghosts (triangle), and high-dimensional neighbors (diamond) in the three datasets. Cases 1 and 3 show that original and ghost projections are separated into two groups. Cases 2 and 4 show that ghost projections are cohesive, while the original is separated from all its ghosts. Case 5 illustrates where both the original and its ghosts are scattered, without forming a clear group.

Table 8: Guidelines for interpreting the original and ghost projections.

Pattern	Interpretation
P1: Original and ghosts converge to a compact region	**Stable.** The projection of the data point is relatively robust to stochasticity, and the location of the original can be trusted.
P2: Original and ghosts are split into two or more distinct groups	**Unstable.** Multiple possible projections exist; the original point represents one of the possible locations. Interpret its location with caution.
P3: Original and ghosts are widely scattered with no clear group	**Highly unstable.** Weak attractive forces cause the projections to be dominated by random repulsive forces. The projections are strongly subject to stochasticity.
P4: Ghosts converge, but the original is separated from the ghost group	**Deceptive original.** The original point may be misleading due to stochastic misplacement. The ghost cluster may offer a more reasonable representation of the point's projection.

categorized common patterns we observed during experiments into four patterns, summarized in Tab. 8.

P1: Original and ghosts converge to a compact region. The original projection and all of its ghosts settle in a compact region, suggesting that the optimization process yields a consistent result despite stochastic perturbations and random negative sampling. The original projection is considered (r,d)-stable if the original and ghost are confined in a circle of radius d, and they can be interpreted with high confidence. This is the most frequent pattern we observed in practice.

P2: Original and ghosts are split into two or more distinct groups. The original projection and its ghosts do not form a single cohesive cluster but instead diverge into multiple distinct regions. While most examples exhibit a split into two primary clusters, we also observed cases where they are dispersed across three or more clusters. This pattern suggests that the original projection is unstable, with multiple possible locations affected by random processes. Therefore, its position should be interpreted with caution. If a single region must be chosen to characterize the corresponding data points, it may be reasonable to select the cluster containing the majority of their ghost projections.

P3: Original and ghosts are widely scattered with no clear group. The original projection and its ghosts are dispersed without forming a clear group. Although less common, this pattern appears frequently in projections lacking clear cluster structures. The dispersion suggests weak attractive forces from neighboring points, allowing repulsive forces from randomly sampled negatives to exert greater influence on their positions. As a result, the original projection becomes highly sensitive to stochastic effects, and no single location can be reliably interpreted as representative.

P4: Ghosts converge, but the original is separated from the ghost group. The ghosts converge tightly to a location that is spatially distant from the original projection. Although rare, this pattern was consistently observed across multiple datasets. In such cases, the original projection is highly unstable and potentially misleading as its location does not reflect the effect of stochasticity if ghosts were not considered.

6 Use Cases

In this section, we present use cases of GhostUMAP2, where we assess the stability of UMAP projections by identifying unstable points. We report five representative cases (Fig. 6) from three real-world datasets drawn from diverse domains: *C. elegans* (biology), Fashion-MNIST (images), and AG News (text). Additional use cases are available in the supplementary material. For all datasets, we used the default hyperparameters and set r and d to 0.1.

The *C. elegans* dataset contains transcriptome profiles of 6,188 single cells collected during *C. elegans* embryogenesis, with each profile in 50 dimensions. We specifically include the *C. elegans* dataset to replicate an analysis workflow commonly used in single-cell transcriptomics. In such analyses [25], UMAP projections are commonly used for clustering cells and assigning cell type annotations to unlabeled cells using known marker genes. However, unstable projections of data points can result in incorrect annotations of cells, thereby compromising the validity of downstream interpretations.

The Fashion-MNIST dataset consists of 70,000 grayscale 28×28 images of 10 classes, such as T-shirts, trousers, and sneakers. We selected this image dataset because it allows for direct visual inspection of samples, helping to interpret the sources of projection instability.

The AG News dataset is a collection of news articles categorized into four labels: World, Sports, Business, and Science/Technology. We sampled 5,000 articles per category and embedded them into 768-dimensional vectors using the DistilBERT model [28]. This dataset was chosen as a representative of text data and is notable for lacking clear separability among categories in the embedding space.

The five use cases are shown in Fig. 6, all captured in GhostExplorer. We show the original projection of an unstable data point that we want to discuss as a cross, its ghosts as triangles, and its neighbors in the original high-dimensional space as diamonds.

Cases 1 and 3 in Fig. 6 show unstable points in the *C. elegans* and Fashion-MNIST datasets, respectively, where the original and ghosts are split into two distinct regions (**P2** in Tab. 8). In Case 1, six ghost points are located along the lower-right branch dominated by the "AFD" cell type, while the original point and the remaining ghosts are positioned near the center of the projection, where cluster boundaries are less clear. Similarly, in Case 3, the original point is located in the "Trouser" cluster, while its ghosts and neighbor points split into groups aligned with both "Trouser" and "Dress." These cases illustrate how cluster membership can be misleading for unstable points, as their projected positions may result from stochasticity rather than meaningful structure.

To gain a deeper understanding of Case 3, we inspected the pixel images of the original point and its high-dimensional neighbors (Fig. 7-A). Five of its neighbors belonged to the same "Dress" class as point (A), while nine belonged to "Trouser". The visual similarity among these neighboring points suggests that the original image is inherently ambiguous. The ghosts are pulled toward both clusters by competing attractive forces, making them particularly susceptible to stochasticity.

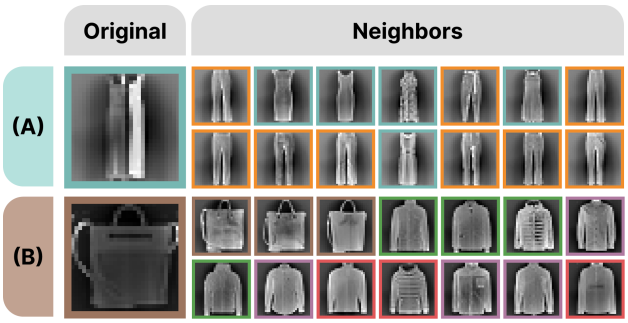

Fig. 7: The corresponding pixel images of two data points (leftmost column) and their high-dimensional neighbors for cases (A) and (B) in Fig. 6. Neighbor images are arranged from top-left to bottom-right in order of increasing distance from the original projection.

Cases 2 and 4 in Fig. 6 represent two unstable data points where ghosts converge, but the original is separated from the ghost group (**P4**). In Case 2, the original projection is near the boundary of unrelated cell types, while its ghosts consistently converge into a branch dominated by the "ASK" (yellow) cell type. Similarly, in Case 4, the original projection locates the data point of "Bag" (brown) within the cluster dominated by the "Pullover" (red), "Coat" (green), and "Shirt" (purple), while its ghosts converge on the correct "Bag" cluster. When examining the images of the original point and its neighbors (Fig. 7-B), we found that three neighbors belonged to the "Bag" class, while the others came from different categories. Both cases demonstrate that the original projection may be a stochastic outcome, leading to incorrect placement, particularly for data points having inconsistent neighbors.

Finally, Case 5 in Fig. 6 illustrates a case where the original projection and ghosts are widely scattered (**P3**). This pattern is frequently observed in datasets where there are weak cluster structures, such as the AG News dataset. The widely spread high-dimensional neighbors may have contributed to this case; when neighbors are dispersed, their attractive forces become diluted, allowing repulsive forces from random negative samples to dominate.

In summary, as demonstrated in Cases 1–5, GhostUMAP2 can identify unstable projections of data points. While we partially explained those unstable points by the dispersion or inconsistency of their high-dimensional neighbors, the reverse does not always hold. For example, stable points not explicitly discussed (e.g., **P1**) often have neighbors scattered in the projection space but remain (r,d)-stable; examples of such cases can be found in the supplementary material. This highlights that GhostUMAP2 offers insights into instability that cannot be inferred from neighbor distributions alone. It is also important to note that class information, represented by point colors and image frames in Fig. 6 and Fig. 7, respectively, is typically unavailable, as dimensionality reduction is an unsupervised method. Even without class labels, our technique can support more reliable interpretations of projections.

7 Discussion and Limitations

7.1 Comparison to GhostUMAP

This work extends a previous study, GhostUMAP [14], by introducing several improvements for measuring instability. First, GhostUMAP2 generalizes GhostUMAP by allowing an arbitrary amount of perturbation to the initial projections of ghost points, in addition to the perturbation applied to the original projection by UMAP. This additional perturbation is controlled by the parameter r, where GhostUMAP corresponds to the special case of $r = 0$. By introducing r, GhostUMAP2 incorporates the stochasticity arising from random initializations more explicitly within the framework.

Second, while GhostUMAP measured instability using the variance among the positions of the original and ghost projections, GhostUMAP2 uses the distance from the original projection to its farthest ghost. This change is motivated by two reasons. On one hand, variance is conceptually misaligned with r, which is defined as a distance metric; measuring instability in terms of distance offers a more interpretable and consistent framework. On the other hand, variance does not give special consideration to the original projection. For example, in several cases (e.g., **P4**), ghost projections form a tight cluster while the original projection lies far apart. Using variance in such cases can be misleading, yielding lower variance, as it treats all projections equally without highlighting the instability of the original point.

The third improvement concerns the acceleration of ghost computations. In previous work, we conservatively reduced the number of points by repeatedly halving the dataset, resulting in a speedup that was limited by the number of halving steps, which was inefficient. In GhostUMAP2, however, we were inspired by the observation that unstable points tend to have larger d_i values. This insight allowed us to adaptively reduce the number of data points based on the degree of instability observed in the middle of the computation, leading to more efficient computation.

7.2 Limitations of GhostUMAP2

While our evaluation and use cases demonstrated the effectiveness of GhostUMAP2 and its adaptive dropping scheme in measuring the instability of points, several challenges warrant further discussion. The first challenge concerns how to address unstable points. Our work focused on identifying unstable points and understanding the reasons behind their instability by analyzing their positional variations. However, we did not explore methods to counteract this instability. The approach in GhostExplorer is passive: it optionally hides unstable points. However, more active approaches could be considered. For example, we can intervene in the optimization process to alter the projections of unstable points. Nevertheless, it is crucial to distinguish between instability caused by stochastic artifacts of DR techniques and the intrinsic ambiguity of the data points themselves. Data points that are intrinsically ambiguous (e.g., Fig. 7-A) may not be accurately represented as a single point in the projection, even though we fix their projections. Investigating the structural or algorithmic causes of instability and handling the identified unstable points accordingly remain future work.

The second challenge involves leveraging findings from GhostUMAP2 to interpret the global structures of projections. While DR techniques are commonly used to explore the global or cluster-level structure of high-dimensional data, analyzing individual ghosts may not be well-suited for this purpose. Addressing this limitation could involve enhanced visual support, such as enabling collective visual analysis of ghost projections. Alternatively, computational approaches could be employed, such as incorporating the ghosts of clusters during optimization. Extending the interpretation and application of ghosts to coarser granularities would be a promising future research direction.

The final challenge is an extension to different DR techniques. While we demonstrated the application of ghosts with UMAP as a concrete example, our (r,d) framework can also be applied to other DR techniques that employ force simulations, such as LargeVis [31] and t-SNE. Investigating pointwise stability across different DR techniques could provide a more consistent basis for analysis and a means to evaluate the stability of techniques.

8 Conclusion

In this work, we present GhostUMAP2 to measure and analyze the instability of point projections, extending UMAP. Using our (r,d)-stability framework, we define a point as unstable if any of its ghosts, perturbed within a radius r in the initial projection, is positioned farther than d in the final projection. Ghost projections are computed through a joint optimization process, treating ghosts as faithful but alternative outcomes of the original points under stochastic conditions. To mitigate the computational cost increased by ghost tracking, we apply an adaptive dropping scheme that achieves up to 2.4× speedup. Through the use cases on three real-world datasets, we identify four instability patterns and demonstrate that GhostUMAP2 enables a deeper understanding of stochasticity by revealing positional alternatives and potential misplacements. By complementing conventional UMAP analysis, GhostUMAP2 offers a practical tool for identifying and interpreting projection variability, promoting a more robust analysis of low-dimensional embeddings.

Acknowledgments

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190421, Artificial Intelligence Graduate School Program (Sungkyunkwan University)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00221186). This work has been supported in part by the Knut and Alice Wallenberg Foundation through Grant KAW 2019.0024.

References

[1] A. Asuncion, D. Newman, et al. UCI machine learning repository, 2007. 6

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, 14, 2001. 7 pages. https://proceedings.neurips.cc/paper_files/paper/2001/file/f106b7f99d2cb30c3db1c3cc0fde9ccb-Paper.pdf. 1

[3] R. G. Brown. *Exponential Smoothing for Predicting Demand*. Philip Morris Records; Master Settlement, 1956. https://www.industrydocuments.ucsf.edu/docs/jzlc0130. 4, 7

[4] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical Japanese literature. 2018. 8 pages. doi: 10.20676/00000341 6

[5] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2153–2173, 2019. doi: 10.1109/TVCG.2019.2944182 2

[6] T. Fujiwara, Y.-H. Kuo, A. Ynnerman, and K.-L. Ma. Feature learning for nonlinear dimensionality reduction toward maximal extraction of hidden patterns. In *Proc. PacificVis*, pp. 122–131, 2023. doi: 10.1109/PacificVis56936.2023.00021 2

[7] G. E. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*, 15, 2002. 8 pages. https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf. 2

[8] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417, 1933. doi: 10.1037/h0071325 2

[9] H. Jeon, A. Cho, J. Jang, S. Lee, J. Hyun, H.-K. Ko, J. Jo, and J. Seo. ZADU: A Python library for evaluating the reliability of dimensionality reduction embeddings. In *Proc. VIS*, pp. 196–200. IEEE, 2023. doi: 10.1109/VIS54172.2023.00048 2

[10] H. Jeon, H.-K. Ko, J. Jo, Y. Kim, and J. Seo. Measuring and explaining the inter-cluster reliability of multidimensional projections. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):551–561, 2022. doi: 10.1109/TVCG.2021.3114833 2

[11] H. Jeon, H.-K. Ko, S. Lee, J. Jo, and J. Seo. Uniform manifold approximation with two-phase optimization. In *Proc. VIS*, pp. 80–84. IEEE, 2022. doi: 10.1109/VIS54862.2022.00025 5

[12] J. Jo, F. Vernier, P. Dragicevic, and J.-D. Fekete. A declarative rendering model for multiclass density maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):470–480, 2019. doi: 10.1109/TVCG.2018.2865141 5

[13] M. Jung, J. Choi, and J. Jo. Projection Ensemble: Visualizing the robust structures of multidimensional projections. In *Proc. VIS*, pp. 46–50. IEEE, 2023. doi: 10.1109/VIS54172.2023.00018 2

[14] M. Jung, T. Fujiwara, and J. Jo. GhostUMAP: Measuring pointwise instability in dimensionality reduction. In *Proc. VIS*, pp. 161–165. IEEE, 2024. doi: 10.1109/VIS55277.2024.00040 1, 4, 5, 6, 9

[15] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. doi: 10.1007/BF02289565 2

[16] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based Python JIT compiler. In *Proc. LLVM*, pp. 1–6, 2015. doi: 10.1145/2833157.2833162 1, 2

[17] Y. LeCun, C. Cortes, and C. J.C. Burges. The MNIST database of handwritten digits. https://www.openml.org/search?type=data&id=554, 1999. Accessed: 2024-3-26. 6

[18] J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443, 2009. doi: 10.1016/j.neucom.2008.12.017 2

[19] S. Lespinats and M. Aupetit. CheckViz: Sanity check and topological clues for linear and non-linear mappings. *Computer Graphics Forum*, 30(1):113–125, 2011. doi: 10.1111/j.1467-8659.2010.01835.x 2

[20] H.-Y. Lu, T. Fujiwara, M.-Y. Chang, Y.-c. Fu, A. Ynnerman, and K.-L. Ma. Visual analytics of multivariate networks with representation learning and composite variable construction. *IEEE Transactions on Visualization and Computer Graphics*, 2024 (Early Access). doi: 10.1109/TVCG.2024.3423728 5

[21] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, 2013. doi: 10.1109/TVCG.2013.65 5

[22] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv:1802.03426*, 2018. doi: 10.48550/arXiv.1802.03426 1, 5

[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 2013. 9 pages. https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf. 1

[24] L. G. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE Transactions on Visualization and Computer Graphics*, 25(8):2650–2673, 2018. doi: 10.1109/TVCG.2018.2846735 2

[25] J. S. Packer, Q. Zhu, C. Huynh, P. Sivaramakrishnan, E. Preston, H. Dueck, D. Stefanik, K. Tan, C. Trapnell, J. Kim, et al. A lineage-resolved molecular atlas of *C. elegans* embryogenesis at single-cell resolution. *Science*, 365(6459):eaax1971, 2019. doi: 10.1126/science.aax1971 6, 8

[26] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008. doi: 10.1109/TVCG.2007.70443 2

[27] J. Rennie. 20 Newsgroups. http://qwone.com/~jason/20Newsgroups/. Accessed: 2024-04-30. 6

[28] V. Sanh. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *Proc. EMC*^{2}, 2019. doi: 10.48550/arXiv.1910.01108 8

[29] M. Sips, B. Neubert, J. P. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *Computer Graphics Forum*, 28(3):831–838, 2009. doi: 10.1111/j.1467-8659.2009.01467.x 2

[30] J. Stahnke, M. Dörk, B. Müller, and A. Thom. Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):629–638, 2016. doi: 10.1109/TVCG.2015.2467717 2

[31] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proc. WWW*, pp. 287–297, 2016. doi: 10.1145/2872427.2883041 1, 5, 9

[32] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale information network embedding. In *Proc. WWW*, pp. 1067–1077, 2015. doi: 10.1145/2736277.274109 1

[33] J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19(6-7):889–899, 2006. doi: 10.1016/j.neunet.2006.05.014 2

[34] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017. doi: 10.48550/arXiv.1708.07747 6

[35] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, 28, 2015. 9 pages. https://proceedings.neurips.cc/paper_files/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf. 6