we summarize by two key questions, as follows:

We present the data for exploration or communication via 2D or 3D (DR) techniques, have a key place in the toolset of data scientists. Practitioners want to apply DR to solve their specific analysis, simplification, prediction, or visualization tasks on their data. Their main question relates to how to find the DR technique that optimally covers their requirements, including adding such algorithms to their data-processing pipeline. Researchers that develop novel DR techniques need to compare their (new) technique with existing ones to demonstrate its added value, and next want to share it with as many other practitioners and researchers as possible.

Having worked in both practitioner and researcher roles in various teams for over 10 years, we have observed several challenges, which we summarize by two key questions, as follows:

- **Q0 (Practitioners):** How to choose the best DR algorithm implementation for my context from the wide set of options available in the public arena?

- **Q0 (Researchers):** How to compare my new DR algorithm against the existing ones, and next share it with as many other practitioners and/or researchers as possible?

Both questions can be answered in many ways, and using many instruments, e.g., surveys, benchmarks, and open-source repositories. At a meta level, we ask ourselves: How to answer these questions efficiently and effectively? For practitioners, the search space (of existing DR algorithm implementations) is huge. How to approach the search process, starting from one’s context-specific requirements, to find as quickly as possible the best DR techniques that fit these requirements? For researchers, the effort of developing new DR techniques is already large. How to ensure, with minimal effort, that the developed techniques are indeed better (and if so, how to measure this) than existing ones?

In this paper, we examine both above questions in a systematic way. We identify the workflows that typical practitioners and researchers follow when answering these questions. Next, we identify available instruments to complete each step of these workflows, and discuss the challenges and limitations we observed when applying these instruments. In particular, we propose an architecture for a benchmark for DR evaluation that is generic and extensible in terms of datasets, DR algorithms, quality metrics, and visualizations. Finally, we discuss ways forward for the community of practitioners and researchers interested in applying, respectively developing, DR algorithms.

## 1. Introduction

Multidimensional projections, also called Dimensionality Reduction (DR) techniques, have a key place in the toolset of data scientists. They support many tasks dealing with high-dimensional data, such as analysis (finding interesting patterns in the data, e.g., clusters, subspaces, or outliers); simplification (reducing the number of dimensions needed to capture the data structure); prediction (classification or regression tasks executed on the data samples); and visualization (presenting the data for exploration or communication via 2D or 3D scatterplots) [HG02, LMW15, KH13, TLZM16].

Different audiences have different requirements for DR algorithms. Practitioners want to apply DR to solve their specific analysis, simplification, prediction, or visualization tasks on their data. Their main question relates to how to find the DR technique implementation that optimally covers their requirements, including adding such algorithms to their data-processing pipeline. Researchers that develop novel DR techniques need to compare their (new) technique with existing ones to demonstrate its added value, and next want to share it with an as wide as possible public.

Having worked in both practitioner and researcher roles in various teams for over 10 years, we have observed several challenges, which we summarize by two key questions, as follows:

- **Q0 (Practitioners):** How to choose the best DR algorithm implementation for my context from the wide set of options available in the public arena?

- **Q0 (Researchers):** How to compare my new DR algorithm against the existing ones, and next share it with as many other practitioners and/or researchers as possible?

Both questions can be answered in many ways, and using many instruments, e.g., surveys, benchmarks, and open-source repositories. At a meta level, we ask ourselves: How to answer these questions efficiently and effectively? For practitioners, the search space (of existing DR algorithm implementations) is huge. How to approach the search process, starting from one’s context-specific requirements, to find as quickly as possible the best DR techniques that fit these requirements? For researchers, the effort of developing new DR techniques is already large. How to ensure, with minimal effort, that the developed techniques are indeed better (and if so, how to measure this) than existing ones?

In this paper, we examine both above questions in a systematic way. We identify the workflows that typical practitioners and researchers follow when answering these questions. Next, we identify available instruments to complete each step of these workflows, and discuss the challenges and limitations we observed when applying these instruments. In particular, we propose an architecture for a benchmark for DR evaluation that is generic and extensible in terms of datasets, DR algorithms, quality metrics, and visualizations. Finally, we discuss ways forward for the community of practitioners and researchers interested in applying, respectively developing, DR algorithms.

## 2. Background

We first introduce a few definitions. A projection technique is a function \( P : \mathbb{R}^n \rightarrow \mathbb{R}^m \), which maps a set \( D \subset \mathbb{R}^n \) of \( n \)-dimensional points to a same-size \( m \)-dimensional scatterplot \( P(D) \subset \mathbb{R}^m \), where typically \( m \in \{2, 3\} \). \( P \) aims to capture the structure of an input
dataset \( D \) by preserving various aspects thereof in \( P(D) \), e.g., inter-
point distances or point neighborhoods. To quantify how well \( P \)
preserves such data aspects, quality metrics [BTK11] are used. An implementa-
tion of a projection technique \( P \) is a reusable software component, e.g. executable or library, that computes the function \( P \).

Two types of requirements come into play when answering \( Q_P \)
and \( Q_R \), as usual when engineering software systems:

- **functional requirements** describe properties of the DR technique it-
  self. These include the type of data the projection accepts (nD sam-
ples or a distance matrix), whether \( P \) is deterministic or stochas-
tic, linear or non-linear, global or local, quality (measured by
quality metrics), computational and memory scalability, and out-
of-sample ability. Such aspects can be usually inferred from the
technique’s description;

- **non-functional requirements** describe properties of implementa-
tions of DR techniques, e.g. ease of use, documentation, portabil-
ity, third-party software components and programming language
needed, and interoperability with other toolkits. Finding these
typically requires one to study specific implementations of \( P \).

The questions \( Q_P \) and \( Q_R \) listed in Sec. 1 have preoccupied both
practitioners and researchers, increasingly more in the last decade,
when many DR techniques have emerged in the literature. Several
sources of information are typically used to answer them. We rank
these sources based on how strongly they support answering \( Q_P \)
and \( Q_R \), on an ordinal scale ranging from ‘-’ (least) to ‘++’ (most), as
follows.

**Papers** (\( Q_R : +++; Q_P : -=- \)): Technical papers describing DR tech-
niques are the primary information source for researchers aiming to
understand and/or extend such techniques. Papers discuss functional
aspects of the presented techniques well, but comparisons with other
techniques are in general limited to a few. Non-functional aspects and
implementation details are less thoroughly touched upon in papers,
which leaves \( Q_P \) largely unanswered.

**Surveys** (\( Q_R : +++; Q_P : + + \)): Surveys compare (tens of) projections
and consider more functional aspects (e.g., metrics) than technical
papers (see e.g. [NA18, vP09, EMK19a, EHH12, SVM14]). Surveys offer a very
good way to choose techniques based on their functional properties.
Yet, surveys rarely discuss how to choose specific imple-
mentations of these techniques, and also discuss less non-functional
aspects.

**Benchmarks** (\( Q_R : --; Q_P : + + \)): Benchmarks gather concrete
datasets and projection/metric implementations to help both practi-
tioners and researchers to compare practically DR techniques against
each other. They also help replicability and are very common in
other fields of computer science, e.g. machine learning [MCC19] or
medical imaging [RKHH99, MFM*13]. However, benchmarks are
rare in the DR community. Three notable recent benchmarks are
Espadoto et al. [EMK19a] (18 datasets, 44 projection techniques,
and 7 metrics); Vernier et al. [VGG18*20] (focus in dynamic DR – 10
datasets, 11 techniques, 12 metrics); and SmallVis [Mel] (focus on
t-SNE, UMAP, and LargeVis [TLZ16]).

**Frameworks** (\( Q_R : --; Q_P : + + \)): Frameworks are collections of
DR technique implementations designed, documented, and coded for
reusability. They come as libraries, e.g. scikit-learn [PGV11], Tap-
kee [LWG13]; and turnkey systems, e.g. MATLAB (which provides
out-of-the-box implementations of PCA, Factor Analysis [J086],
NMF [LS01], MDS [Tel58, Kn06] and t-SNE [vH08], Projection-
Explorer [JCC11], and VisPipeline [vis], which provide end-to-end
tools for interactive exploration of projections. An extreme model
of frameworks are code bases that implement a single technique,
e.g. Van der Maaten’s t-SNE [vH08], dt-SNE [RFT16], and
Espadoto’s deep-learning DR technique [EHT19]. While frameworks
best support \( Q_R \), finding which implementations (in which frame-
works) best match a practitioner’s set of requirements still requires
significant manual trial-and-error testing of the DR implementations
they provide.

From the above, we see that the search space for \( Q_P \) and \( Q_R \) is
large and heterogeneous structured. This affects both practitioners
– it is not evident where to start searching, and how to systematically
search; and researchers – there’s no unanimously accepted answer to
what to compare against, what to compare on, how to compare, and
how to report the results; also, researchers face the question on how
to best share their results with others, e.g., simply release their code
or take the time to integrate it with some framework. Hence, we find
a salient gap between DR research and practice. Our aim next is to
provide insights on how to fill this gap with a low effort, and where
the largest unanswered challenges reside.

3. Operational Workflows

We identify two related, but not identical, workflows that practi-
tioners, respectively researchers, follow to answer their respective
questions \( Q_P \) and \( Q_R \) stated in Sec. 1. We inferred these workflows
both from studies of existing papers and surveys in the DR literature,
and from our own experience with answering both \( Q_P \) and \( Q_R \) in
practice. Concerning our experience, we have studied 46 actual DR
implementations. These implementations, and their main functional
and non-functional aspects, are listed in Tab. 1. Additional func-
tional aspects of these techniques, such as complexity and quality are
provided in recent surveys [EMK19a, VGD18*20].

Figure 1 depicts the steps of both these workflows, which we
detail next. Colored dots in the middle table show which informa-
tion sources (surveys, papers, benchmarks, or our own analysis in
Tab. 1) are mainly used to support every workflow step. For each
step, we next discuss the main questions asked by practitioners and
researchers, outline how these can be answered, and which are the
challenges we observed in doing this.

3.1. Practitioner workflow

A. Search techniques. Where do I start searching? Starting points for
this search are, obviously, technical papers on DR techniques and,
more broadly, surveys thereof [EMK19a, NA18]. Additional
search sources are conference presentations, blogs, and peer input.
This search typically delivers a (large) subset of potentially suitable
DR technique candidates (but usually no specific implementations).

B. Search implementations. Where do I find implementations? Ta-
ble 1 provides our survey of available implementations, with frame-
works providing these in Tab. 2. Related to the search process, we
observed several points. Since many data science projects use Python,
practitioners would likely favor Python-based DR implementations.
We found out that scikit-learn [PGV11] provides high-quality im-
plementations of many well-known DR techniques. However, other
techniques come in different languages (Tapkee [LWG13]; C++,
Van Der Maaten’s DR Toolbox [VdMPvdH17]: MATLAB; and
Vispipeline [vis]: Java). These require manual Python wrapping,
which is not easy for the average user, and may hamper adoption of
less known DR techniques, which are not available anywhere else.

C. Select implementations. How do I choose this selection? This involves
considering both functional and non-functional requirements.
Table 1: Studied DR technique implementations with their functional and non-functional aspects listed.

<table>
<thead>
<tr>
<th>Projection</th>
<th>Linearity</th>
<th>Input</th>
<th>Neighborhood</th>
<th>Parameters</th>
<th>Out-of-sample</th>
<th>Deterministic</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNProj</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>1</td>
<td>no</td>
<td>yes</td>
<td>Keras</td>
</tr>
<tr>
<td>Diffusion Maps [EHT19]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>Tapkee</td>
</tr>
<tr>
<td>Factor Analysis [Jol86]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Fastmap [FJR95]</td>
<td>NL</td>
<td>D</td>
<td>G</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>GDA [BAS06]</td>
<td>NL</td>
<td>D</td>
<td>G</td>
<td>1</td>
<td>no</td>
<td>yes</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>GPLVM [Law04]</td>
<td>LIN</td>
<td>D</td>
<td>G</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>Fast ICA [Hyv09]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>2</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>IDMAP [MPL06]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>Isomap [TOL00]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Landmark Isomap [CCG06]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>LAMP [CC14]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>Laplacian Eigenmaps [BN02]</td>
<td>NL</td>
<td>L</td>
<td>D</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>LLE [TB02]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>no</td>
<td>yes</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>LLE [R956]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Hessian ILE [DG01]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Modified ILE [ZW97]</td>
<td>NL</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>LMNN [WBS06]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>no</td>
<td>yes</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>LPP [BN04]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>Tapkee</td>
</tr>
<tr>
<td>LSP [FNSL08]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>4</td>
<td>yes</td>
<td>no</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>LTKA [ZZH04]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Linear ILETA [ZY2007]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>Tapkee</td>
</tr>
<tr>
<td>Manifold Charting [Bau02]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>MCLM [GB08]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>MDS [Tur58]</td>
<td>NL</td>
<td>D</td>
<td>G</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Landmark MDS [DIT04]</td>
<td>NL</td>
<td>D</td>
<td>G</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Nonmetric MDS [Kra64]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Landmark MVO [WPS05]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>NMF [EL09]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>PBC [P060]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>PCA [Jol86]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Incremental PCA [ELLY08]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Kernel PCA (polynomial) [SMH98]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Kernel PCA (RBF) [SMH98]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Kernel PCA (Sigmoid) [SMH98]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Probabilistic PCA [TB99]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>DR Toolbox</td>
</tr>
<tr>
<td>Sparse PCA [ZIT06]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>PLSP [IEP11]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>Vispipeline</td>
</tr>
<tr>
<td>Random Projection (Gaussian) [Das00]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Random Projection (Sparse) [Das00]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>no</td>
<td>yes</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>Rapid Simulation [FokDK09]</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>t-SNE [vH08]</td>
<td>NL</td>
<td>D</td>
<td>L</td>
<td>3</td>
<td>no</td>
<td>no</td>
<td>Multicore TSNE, t-SNE archive, TFJS-t-SNE</td>
</tr>
<tr>
<td>t-SNE repository</td>
<td>NL</td>
<td>S</td>
<td>G</td>
<td>0</td>
<td>no</td>
<td>no</td>
<td>Tapkee</td>
</tr>
<tr>
<td>Truncated SVD [BMT09]</td>
<td>LIN</td>
<td>S</td>
<td>G</td>
<td>1</td>
<td>yes</td>
<td>no</td>
<td>scikit-learn</td>
</tr>
<tr>
<td>UMAP [MBM08]</td>
<td>NL</td>
<td>D</td>
<td>L</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>UMAP library</td>
</tr>
<tr>
<td>dt-SNE [RTT16]</td>
<td>NL</td>
<td>D</td>
<td>L</td>
<td>4</td>
<td>no</td>
<td>no</td>
<td>dt-SNE repository</td>
</tr>
<tr>
<td>NSnSp路演 [EIT19]</td>
<td>LIN</td>
<td>S</td>
<td>L</td>
<td>network size</td>
<td>yes</td>
<td>yes</td>
<td>NNProj code</td>
</tr>
</tbody>
</table>

Table 2: DR frameworks used in the study.

<table>
<thead>
<tr>
<th>Framework name</th>
<th>Available at</th>
<th>Programming Language(s)</th>
<th>Documentation (Quality)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR Toolbox</td>
<td>[link]</td>
<td>MATLAB Python and C++</td>
<td></td>
</tr>
<tr>
<td>Multicore TSNE</td>
<td>[link]</td>
<td>Python and C++</td>
<td></td>
</tr>
<tr>
<td>scikit-learn</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Tapkee</td>
<td>[link]</td>
<td>Keras</td>
<td></td>
</tr>
<tr>
<td>umap-learn</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Vispipeline</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>t-SNE archive</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>UMAP repository</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>TFJS-t-SNE</td>
<td>[link]</td>
<td>Python</td>
<td></td>
</tr>
</tbody>
</table>

former, one can easily screen DR techniques based on their properties listed in recent surveys [EMK*19a, NA18]. For the latter, we point to our own survey in Tabs. 1 and 2. Based on both requirement types, a ranking is made and a subset of candidates are selected. We observe several points concerning non-functional requirements. Regarding documentation, outside of scikit-learn and UMAP, which are well documented, most libraries we found had little to no documentation at all (Tab. 2), making adoption hard. In some cases, reading the source code is the only option, like the case of many techniques found in Van Der Maaten’s DR Toolbox. Even in the cases where the library is well documented, we often found not enough details on the role of each parameter in the final quality of the projection, and even less on the interaction between parameters. Separately, the API design promoted by DR libraries can vary enormously. Each library has its own conventions on data format and parameters, which makes the problem of interfacing hard. Take, for example, the example below, which runs t-SNE [vH08] on some dataset X using scikit-learn and Tapkee (Listings 1 and 2). We believe the two-step API of scikit-learn (create object with parameters, call fit_transform()) to be much simpler to understand for the average user than Tapkee’s method of chaining with globally-namespaced, non-specific keywords.

**Listing 1:** t-SNE example with scikit-learn

```python
from sklearn.manifold import TSNE
tse = TSNE(perplexity=30)
output = tse.fit_transform(X)
```

**Listing 2:** t-SNE example with Tapkee

```python
tapkee = Tapkee()
tapkeeOutput tsne = initialize()
 .withParameters((
    method=1DistributedStochasticNeighborEmbedding, 
    target_dimension=2, 
    s_ne_perplexity=30))
```
Figure 1: Workflows for selecting DR implementations for practitioners (top, see Sec. 3.1) and researchers (bottom, see Sec. 3.2). Colored dots indicate which input data (surveys, papers, benchmarks, Tab. 1) are used by the two roles in each step.

D. Test implementation. How to test the selected implementations?

Surveys and code inspection cannot tell everything about a specific DR implementation. One needs to actually test an implementation on given datasets and against a set of metrics. For this, benchmarks are needed. To our knowledge, only three such benchmarks exist in DR landscape [EMK*19a, VGdS*20, Mel]. Yet, such benchmarks are never complete, so they need to be extended with additional DR implementations, datasets, and metrics. We discuss how such benchmarks can be designed for extensibility next in Sec. 4.

3.2. Researcher workflow

A. Search techniques. This step is largely similar to step A for practitioners (Sec. 3.1). The focus is though different for researchers, who are mainly interested in finding functional limitations of existing DR techniques that they want to improve upon, rather than the non-functional ones that are more relevant to practitioners.

B. Implement own technique. This step can proceed, for the most part, independently from existing DR implementations. However, some researchers may choose to follow coding standards and API conventions of existing (successful) DR implementations to already maximize exposure at this stage.

C. Test implementation. This step typically uses the same benchmarks as in step D for practitioners. An important part of this step is presenting the test results. While less important for practitioners, researchers need compelling ways to show that their (novel) techniques perform well vs many other techniques on many metrics to convince their peer researchers. For this, metric visualizations are used, the most prominent being bar and boxplot charts [vP09], tables [NA18], and space-filling charts [EMK*19a]. A challenging aspect here is that the space to visualize is at least four-dimensional (datasets, metrics, DR techniques, parameter settings). Ideally, creating such visualizations, and adding new visualizations, should be supported by the benchmarks.

D. Share implementation. Once a DR implementation has been successfully tested, its further impact critically depends on how easily it is shared with practitioners. Doing this follows two approaches. Standalone code is the easiest way – the researcher just makes her DR implementation available via a website or software repository. Examples hereof are dt-SNE, NNproj, and DR Toolbox. One issue with this approach is that integrating DR code having non-standard APIs with other components of a data science pipeline can be hard. Also, standalone code is less visible to practitioners than code shared via well-known frameworks (discussed next). Finally, adaptive maintenance is less often done on standalone code, which can easily break it upon the evolution of third-party components it uses. An example hereof is dt-SNE, which depends on the unmaintained Theano [the] library. However, good examples of standalone code sharing exist, such as t-SNE archive and UMAP (Tab. 2), discussed below. Framework integration, the second approach, adds the DR implementation to a mainstream data science or similar framework. Examples hereof are scikit-learn, Tapkee, MATLAB, and Vispipeline. This takes considerably more effort than sharing standalone code, as the code to integrate must comply with framework APIs and documentation constraints, but favors (far) larger exposure. Framework integration can be hard for DR techniques which need more than data input-output communication with the framework. Examples hereof are projections which advertise landmarks interactively placed by the user, such as LAMP [JCC*11] or for techniques which use GPU acceleration, e.g. TFJS-t-SNE [PTM*19]. To integrate these, a framework should provide APIs for interaction, respectively for GPU computing.

Standalone code sharing has limitations, but can still be very successful. Two good examples are UMAP [MHH18] and t-SNE...
Van der Maaten) chose to provide a ‘portal’ for implementations in several languages (C++, Python, JavaScript, CUDA, R, Java, MAFLAB), thereby simplifying integration with third-party code in all these languages.

4. Architecting an Evaluation Benchmark

Having a benchmark to quantitatively evaluate DR algorithms is essential for both the practitioner and researcher workflows (see Sec. 3). As mentioned there, not many such benchmarks exist. Importantly, by a benchmark, we do not understand here just a ‘passive’ collection of high-dimensional datasets to battle-test DR techniques, but rather a **runnable** software system that lets one select datasets, DR technique implementations (and their parameter values), metrics, execute them, and visually inspect the results in an easy and highly automated way. The only two benchmarks that approach this definition are [EMK19a] (for static DR techniques) and [VGdS20] (for dynamic DR techniques). However, these benchmarks have their own limitations. Extending them involves, at points, manually reading and reverse-engineering their code, which is hard. Creating an even better (broader, easier to use) benchmark is a high-effort task involving many decisions.

We aim to support the interested users in either the extension task or the design-from-scratch task by proposing a generic architectural template for such a benchmark (see Fig. 2). We created this architecture by studying the two benchmarks [EMK19a, VGdS20], including the implementation [EMK19b] of the former, and next unifying their design and implementation, aiming to generalize and simplify. We believe that our proposal meets well the generivity and extensibility requirements, as detailed next.

**Overall design:** The benchmark follows a dataflow execution model (**see execute** function in Fig. 2). Datasets \(d\) from a database \(D\) are projected in turn by several DR technique implementations \(p\) from a DR technique collection \(P\), using several parameter values \(params\), yielding corresponding 2D scatterplots \(d_{2D}(p, params)\). For each such scatterplot, a set of projection quality metrics \(m(d, d_{2D})\) is computed. The results \(d_{2D} and \(m\) are stored in a result database \(R\), implemented using the Python “pickle” binary format files for efficiency. These results can be next visually explored by visualizations selected by the analyst from a given set \(V\). We next detail each of the main components outlined above.

**Dataset collection:** This is the set of datasets to be considered in the evaluation, stored as a name-value dictionary \((name, d_{data})\). The values \(d_{data}\) are URLs pointing to actual files that store the data samples \(x_1, \ldots, x_n\), each having \(n\) dimensions \(x_1, \ldots, x^n\), in a table format, using either CSV following the ‘tidy data’ [Wic14] standard or binary NumPy [WCV11].

**Projection techniques:** This is the set of DR technique implementations to be evaluated, as well as their parameters to be used during evaluation, stored as a set of tuples \((name, p_{code}, p_{params})\). Here, \(p_{code}\) points to the Python implementation of a DR technique, which is a function that expects a dataset \(d\) and parameter-set \(params\) and returns the computed 2D scatterplot \(d_{2D}(params)\). A so-called **parameter grid**, i.e. a table having as many columns as parameters \(p\) expects, and one row per parameter-set to be used during the evaluation. For instance, if we want to evaluate a t-SNE implementation of \(p\) that expects two parameters \(perplexity\) and number of **iterations** (see [vH08] for details on these parameters), which range in **perplexity** \(\in [20, 40]\) and **iterations** \(\in [100, 150, 200]\), we provide a parameter-grid \(p_{grid}\) table equal to the Cartesian product \(\{20, 40\} \times \{100, 150, 200\}\). This design allows flexibly evaluating DR techniques having different numbers and types of parameters over user-supplied parameter grids.

**Metrics \(M\):** This is the set of quality metrics to be used to assess the benchmarked projections, stored as a dictionary \((m_{name}, m_{code})\). Here, \(m_{code}\) points to the Python implementation of a metric, which is a function that expects a dataset \(d\) and its 2D projection \(d_{2D}\) and computes one of the predefined values \(v\), then the visualization will generate separate small-multiples for each of the values of that parameter in \(R\). If a parameter is set to the predefined value \(v\), then the visualization will generate a single plot for all values of that parameter. These three options are conceptually analogous to the SQL operations **SELECT**, **WHERE**, and **SELECT, SUM** features. This allows one to easily specify visualizations having different levels of data aggregation. For example, if we want to display quality metrics, setting \(m_{name} = each, m_{code} = each\) creates one separate metric plot for each dataset and DR technique in \(R\). Setting \(m_{name} = each, m_{code} = aggregate\) creates one metric plot that shows, for each dataset, the aggregate (e.g., average, depending on the actual Vis implementation) values of metrics over all DR techniques. Finally, setting a parameter to a specific value, e.g. \(m_{name} = t-SNE\), creates a visualization only for the respective DR technique entries present in \(R\). The dictionary keys \(m_{name}, m_{code}\) show now their purpose: They are used both for the user to select specific entries in \(R\) to visualize and to create labels in the generated visualizations. This design allows specifying a quite large range of visualizations, see the examples in [EMK19a, VGdS20].

In terms of implementation, actual visualizations can be coded as Python scripts calling Matplotlib [Hun07] (as in [EMK19a]). A more interactive and flexible development workflow can be achieved by using Jupyter notebooks that allow for independent and interactive execution of their code cells and rich presentation of their output (visualizations, narrative text, mathematical equations, tables). Execution automation is supported by Papernim [nap] which allows one to perform parameterization, instantiation, and execution of Jupyter notebooks. For example, to generate for each dataset \(d\) a separate video showing a small-multiple display of all its time-dependent projections \(p(d)\) for \(d \in D\), one can write a template video generation notebook and use Papernim to instantiate a new notebook for each dataset \(d\).

**Extensibility and genericity:** The above architecture is easily extensible: Adding new datasets, metrics, DR techniques, or parameter grids implies simply adding entries to the respective dictionaries. Dictionaries can be implemented either in Python or, even simpler, as folders having filenames as keys and the respective file contents as actual values. For large benchmarks, implementations using relational databases (e.g., SQL) could also be done relatively easily following the same template architecture. The architecture is also generic, since the formats of datasets, respectively signatures of functions implementing DR techniques, metrics, and visualizations, can...
accommodate most, if not all, concrete instantiations thereof that we know of from the DR literature and practice. For instance, all DR technique implementations in Tab. 1 fit this architecture. Parallelization can also be easily incorporated, e.g., by simple running of multiple processes at the level of the for-loops in execute (a design used in [EMK∗ 19a]). Finally, supporting time-dependent DR techniques implies only a small change to the architecture, namely having \( d_{\text{data}} \) point to a sequence of tables rather than a single one (as done in [VGdS∗ 20]).

5. Discussion

Summarizing, we see a number of open challenges to the selection and sharing of DR algorithms that, jointly, create a gap between the state-of-the-art work in DR literature and practical realities in the field, as follows.

Implementation: For selection (adoption), we note the lack of analysis of DR implementations with respect to non-functional requirements such as programming language, documentation quality, and ease of use. These make practitioners stay away from techniques whose implementations score poorly along these requirements and, conversely, favor techniques that have good-scoring implementations. For sharing, we see that there is no single framework that provides implementations of most DR techniques known in the literature – the closest to this is scikit-learn which implements roughly half of the DR techniques for which we found a mainstream implementation. For sharing, we do not see yet a momentum for researchers to develop DR algorithms within mainstream data science frameworks – the dominant sharing form is still standalone code.

Benchmarks – Datasets: Selecting a representative collection of datasets to gauge DR techniques is hard. Typically, papers, surveys, and benchmarks use datasets that are known in DR literature (for historical reasons) or in a given application domain. However, gauging the quality of a DR technique at large should use datasets that ideally represent well any problem. Espadoto et al. [EMK∗ 19a] do a first attempt in this direction by characterizing datasets by traits (e.g., dimensionality, intrinsic dimensionality, sparsity, provenance) and create a benchmark by sampling these dimensions. Vernier et al. [VGdS∗ 20] use the same idea and aim to also cover dataset dynamics. However, both these surveys admit to only very sparsely sample the space of all possible datasets. Coming up with a good such sampling is an open, and important problem, in DR practice.

Benchmarks – Techniques: To our knowledge, most DR surveys and benchmarks focus on techniques that handle quantitative data, and static projections (except [VGdS∗ 20]). Adding DR techniques that handle other attribute types such as categorical is another open direction towards creating comprehensive benchmarks.

Benchmarks – Metrics: DR literature knows tens of different quality metrics [BTK11]. However, existing benchmarks implement only very few – the most being, to our knowledge, the 6 metrics in [EMK∗ 19a]. A benchmark with a wide set of readily-implemented metrics would be of high value to both practitioners and researchers.

Benchmarks – Extensibility: Some DR benchmarks [EMK∗ 19a, EMK∗ 19a] provide code that allows the experiments to be reproduced and allow for some extensibility, in terms of adding new projection techniques, metrics and datasets. In terms of storing results and creating visualizations, users would benefit from a more structured approach, with data saved in portable formats, and with some form of integration with popular data visualization and exploration tools, such as Tableau [CSH03].

6. Conclusions

We presented an overview of the challenges that exist in the process of selecting and sharing DR techniques from the point of view of different audiences, practitioners and researchers, where we described the typical workflows used in their processes. We listed and ranked the most common sources of information about DR techniques, namely, papers, surveys, benchmarks, and frameworks, and compiled a list with popular frameworks and techniques. We described the architecture used in two recent benchmarks and showed how these can be extended to consider more datasets, techniques, and metrics.

We believe the visualization community would benefit from a more integrated, well-documented benchmark framework, where new techniques, datasets, and metrics could be easily added by the user with minimal programming, and with the capability of integration with existing visualization tools.

References

Figure 2: Proposed benchmark architecture and its dataflow execution workflow (Sec. 4).
[Mel] Melville J.: SmallVis benchmark. github.com/jmelville/smallvis. 2, 4


[pap] Papermill. papermill.readthedocs.io. 5


[the] Theano library. deeplearning.net/software/theano. 4


[vis] Visipeline. vimg.ics.uci.edu/viog/tool/1/projection-explorer-pex. 2


[ZYZG07] Zhang T., Yang J., Zhao D., Ge X.: Linear local tangent space alignment and application to face recognition. Neurocomputing 70, 7-9 (2007), 1547–1553. 3


(c) 2020 The Author(s)