S4: Self-Supervised learning of Spatiotemporal Similarity

Gleb Tkachev, Steffen Frey, Thomas Ertl

Abstract—We introduce an ML-driven approach that enables interactive example-based queries for similar behavior in ensembles of spatiotemporal scientific data. This addresses an important use case in the visual exploration of simulation and experimental data, where data is often large, unlabeled and has no meaningful similarity measures available. We exploit the fact that nearby locations often exhibit similar behavior and train a Siamese Neural Network in a self-supervised fashion, learning an expressive latent space for spatiotemporal behavior. This space can be used to find similar behavior with just a few user-provided examples. We evaluate this approach on several ensemble datasets and compare with multiple existing methods, showing both qualitative and quantitative results.

Index Terms—Spatiotemporal Data, Machine Learning, Ensemble Visualization, Visual Exploration

1 Introduction

Advances in simulation methods, compute hardware and measurement technology lead to the generation of spatiotemporal data at an unprecedented rate. In many cases, scientists collect data not from just a single run but from hundreds or even thousands of different configurations. These so-called ensembles are fundamental in studying the effects of varying input parameter values, boundary conditions, materials, etc. (e.g., [1]). Visualization is crucial in making sense of this data. And explorative analysis plays a particularly important role, since there is often only limited prior knowledge of the newly generated data.

In this paper, we focus on the task of identifying other occurrences of an event or a process discovered somewhere in the data. This task is fundamental for various visual analysis scenarios, yet poses significant challenges when unknown events or new types of data need to be investigated. In some cases, there are domain-specific measures of similarity or indicators for specific processes. However, these are not generally available, and the development of such algorithms requires not only detailed domain knowledge but also a lot of time. Therefore, generic techniques are needed to support the interactive exploration, especially when new types of data are involved. Standard metrics operating directly on the raw data representation like the mean squared error (MSE) or the earth mover's distance (EMD) are generic, but do not yield expressive results for most application contexts. Instead, we use a deep learning model that is able to extract more abstract features from the data, allowing it to yield more meaningful results. Learning-based approaches have been shown to deliver great results in the quantification of image similarity [2]. However, such models are often pre-trained on real-world images and are not suitable for scientific data. And training new models on more appropriate data requires labels at some point in the process, which are typically not available for simulation and experimental data.

In this work, we address this challenge and present a data-driven approach to assess the similarity of regions in spatiotemporal scientific data, supporting meaningful query-based visual exploration. Instead of relying on labeled data, we exploit data continuity and assume that nearby locations often contain similar behavior. Combining that with ideas from self-supervised and similarity learning, we set up a task to train a neural network, which is then used to produce an expressive similarity metric for the provided data. With this metric, we enable search for similar behavior in ensemble data, guided by user-constructed queries. Moreover, our efficient implementation and a prototype user interface allow the search to be performed interactively. In summary, we consider the main contributions of this work as follows.

- We propose a self-supervised method of learning a similarity metric for spatiotemporal regions in scientific data to support expressive similarity queries.
- We explicitly enable user interaction by supporting multi-example queries and providing an efficient implementation as well as a prototype user interface for visual exploration.
- We evaluate our approach both qualitatively with a domain expert and quantitatively by comparing to several baselines on both simulation and experimental ensemble data.

- *Gleb Tkachev and Thomas Ertl are with the University of Stuttgart Email: {gleb.tkachev, thomas.ertl}@visus.uni-stuttgart.de*
- *Steffen Frey is with the University of Groningen Email: s.d.frey@rug.nl*

2 Related Work

Ensemble visualization and similarity measures. The analysis of data ensembles is a challenging visualization task [3]. Potter et al. [4] and Sanyal et al. [5] proposed some of the first approaches, studying climate ensembles, while Waser et al. [6] demonstrated a system for interactive steering of simulation ensembles. Sedlmair et al. [7] and Wang et al. [1] provided detailed surveys of the techniques in the area.

In the context of ensemble visualization, a similarity measure often plays an important role, and so many approaches have been proposed. Bruckner and Moeller [8] used squared differences to explore the visual effects simulation space, Hummel et al. [9] determined similarity between regions via joint variance, Wei et al. [10] efficiently computed similarity between local histograms and Kumpf et al. [11] tracked statistically-coherent regions using optical flow. Jarema et al. [12] utilized Gaussian Mixture Models to compute a similarity matrix for vector fields. And Wang et al. [13] proposed a vector field similarity based on a 3D SIFT implementation. Hao et al. [14] constructed octrees to calculate shape similarities for particle data, while He et al. [15] employed surface density estimates for distances between surfaces. Fofonov et al. [16] developed an isosurface-based similarity for multi-fields. We also propose a similarity metric that can be used

for spatiotemporal ensemble data; however, we use a learning-based approach that is domain-agnostic but capable of adapting to the dataset. Furthermore, we focus on the search for similar behavior, allowing the user to interactively influence the similarity score.

Video object detection. The problem of spatiotemporal similarity is related to object detection in video, which is extensively studied in computer vision [17], [18]. Nevertheless, scientific data presents a unique set of challenges. Some detection approaches target specific object categories such as people in the subfield of person re-identification (e.g. [19], [20], [21]), while in our domain-agnostic setting we cannot make similar assumptions. Other, especially deep-learning-based techniques (e.g. [22], [23], [24]), can detect a diverse but fixed set of objects, requiring at least some supervised data. Finally, the computer vision techniques detect spatial objects in video by exploiting frame coherence, while we are interested in fundamentally temporal processes, thus treating temporal and spatial dimensions equally.

Self-supervised learning. In the last years, self-supervised learning has been gaining popularity, especially in computer vision. Dosovitskiy et al. [25] used random image transformations to generate surrogate image classes and learn a robust feature space. Misra et al. [26] learned their representation by predicting if a sequence of video frames was given in the correct order, while Doersch et al. [27] predicted the relative position of two image patches. Continuing the trend, many other self-supervised tasks were proposed in the following years [28]. Our approach also uses the idea of self-supervised learning, but instead of fine-tuning the pretrained model on supervised data, we use the learned representation directly to find similar behavior.

ML in scientific visualization. Machine learning has long been considered to have great potential in visualization [29]. Originally, it found most applications in the area of Visual Analytics (survey in Endert et al. [30]). However, in recent years, several learning-based approaches have been applied in scientific visualization. Some have focused on efficiently representing and interpolating volume data. For example, Zhou et al. [31] presented a CNN-based approach to upscaling volume data, while Han and Wang [32] employed a recurrent generative model to perform interpolation of temporal volume data. Other works aim to learn the volume rendering function. Berger et al. [33] developed a neural network to generate and explore volume-rendered images. Similarly, Hong et al. [34] utilized an adversarial framework to restyle and generate new renderings from existing images. The method by He et al. [35] is related to both groups, as they learn to generate volume rendering images under interpolation of both visual and simulation parameters. The above works focus primarily on the rendering pipeline, while we aim to learn a feature space that supports exploration and navigation of the data. Another use of adversarial methods was presented by He et al. [36], performing comparison for collections of ensembles that represent different simulation models. Han et al. [37] have shown that autoencoders can be used to learn a latent space for the analysis of streamlines and streamsurfaces. Recently, Guo et al. [38] used LSTM autoencoders with attention to embed and find similarities in sequential medical data. The autoencoder objective is related to self-supervised learning and thus to our method. However, we develop a new self-supervised task suitable for similarity searches and focus on staying domain-agnostic, handling varied ensembles of spatiotemporal volume data. Another implicitly self-supervised approach was proposed by Tkachev et al. [39], who used temporal prediction with CNNs to detect irregular behavior as well as estimate ensemble similarity. We propose a different method for self-supervised ensemble similarity, which is more scalable and allows the user to control the similarity score.

3 Motivation and Design Decisions

Our goal is to support the interactive exploration of ensemble data by developing a similarity metric for spatiotemporal regions. We are particularly interested in a metric that is data- and domain-agnostic, because this has the potential to effectively decouple visualization techniques from the underlying domain-specific aspects. Such a metric could be used in other visualization applications, e.g., to cluster regions, select representative examples or compute projections. In this paper, we focus on searching for occurrences of similar behavior. This is a common task in a large variety of analysis scenarios, and it enables us to directly demonstrate the properties of the learned metric.

We are using a deep learning model to construct our metric because it allows us to make few domain-specific assumptions while still adjusting to the dataset at hand. This ability to adapt to the dataset is the crucial property that leads to a similarity metric more meaningful than a basic distance like MSE (element-wise) or EMD (distribution-based). Furthermore, deep learning models have demonstrated their effectiveness in many related scenarios and most recently also in the context of scientific visualization [32], [33], [34], [35], [39].

Despite their effectiveness, there is a fundamental challenge: the lack of supervised spatiotemporal data in the analysis of simulations and experiments. Indeed, it would be cumbersome and unrealistic for a domain scientist to meticulously label every region of their dataset, which can have high resolution in space and time, as well as numerous members in the case of ensemble data. This would completely defeat the purpose of a domain-agnostic visualization system. To overcome this problem, we propose to use self-supervised learning [25], [26], [27], [28]. The motivation for self-supervised learning comes from the fact that unlabeled data is unusable by typical supervised models, and yet most of the data is unlabeled. How can we utilize this large unlabeled data to improve our models? The answer of self-supervised learning is to define an artificial task (the *pretext* task) on unsupervised data that does not require manual labels. For example, predicting if two images are transformed versions of the same original image to learn transformation-invariant features. We can then train a supervised model on the pretext task and use it to help solve the target task, typically by fine-tuning it on a much smaller labeled dataset [28]. However, we do not have any labeled data available. Instead, we directly use the feature space (representation) learned by the model and compute distances in that space as our similarity metric. To make sure that these distances correspond to similarity, we use the siamese architecture for our model.

Siamese networks [40], [41] are a method of learning similarity. They consist of two identical sub-networks that are joined at the output. Each sub-network takes an input (e.g., an image) and encodes it into a feature space. Then, a distance is computed between the pair of encodings to output their similarity. This setup allows us to train the sub-network (which we call *the encoder*) and also encourages a feature space with useful distances, since we explicitly constrain the model to rely on distances when solving the task. Typically, a siamese model is trained on a labeled dataset of known similarities, but we do not have such data and instead train the model on a self-supervised pretext task.

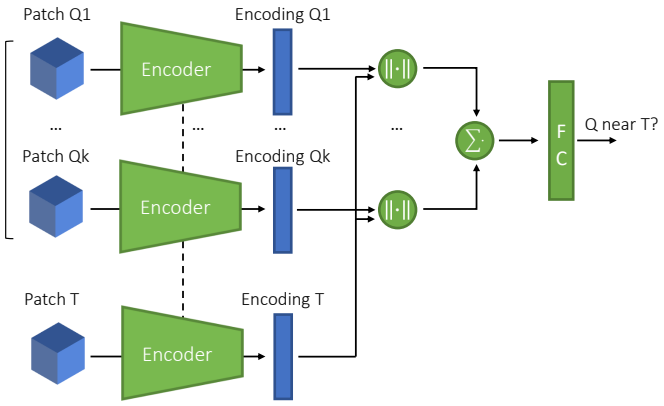


Fig. 1: Our model architecture. We encode the query (Q1, ..., Qk) and the test (T) patches using a convolutional encoder. Then L1 distances from each encoded query patch to the encoded test patch are computed. Finally, the average distance is used by the fully-connected layers (FC) to predict if the query and the test patches are located nearby.

The choice of the pretext task is critical to the learned representation and thus the final performance of our method. Most importantly, we need a task that is as similar as possible to our target task because similar tasks require similar information to be encoded by the model. One should also consider the invariances imposed by the task on the representation, as they will determine what information gets preserved or discarded. With this in mind, we designed our task to be a binary classification problem: "given two spatiotemporal regions of the data, are they nearby in space and time?". More concretely, the model is provided with two rectangular spatiotemporal patches (boxes) from the data and needs to determine whether they originate from the same ensemble member and within a certain spatiotemporal window from each other. This task is closely related to detecting similar behavior because nearby locations often contain similar behavior. Additionally, we explicitly encourage temporal and translational invariance of our representation, which helps detect similar processes that are not perfectly aligned. Note that other formulations are possible, e.g., predicting the distance of patches in space/time (regression) or using contrastive loss, but we prefer a classification formulation for practical reasons. It is simple and produces an intuitive performance metric – accuracy, which allows us to make sure that the model is solving the pretext task during training. In contrast, raw MSE or cross-entropy values would be more difficult to interpret.

The result of the self-supervised training of our siamese model is the encoder that enables us to compute distances between data patches in the learned feature space. Of course, one cannot expect this distance to provide a perfect solution for the target task of finding similar behavior, but we demonstrate below that it is meaningful and is a significant improvement over distances computed directly on the raw data (Sec. 6, Sec. 7 & Sec. 8).

We could apply our technique to any type of a spatiotemporal dataset that is large enough for training, but we focus on ensemble data because it allows us to most effectively demonstrate the utility of our learned similarity. Ensemble datasets often contain many distinct types of behavior but are typically too large to manually search through. We will demonstrate that our approach can detect these behavior types without any domain-specific assumptions. More specifically, we describe and evaluate our approach with 2D+T ensemble data as it enables us to effectively present our results, but we also show that our approach works with 3D+T data (Sec. 7.3). As mentioned above, our similarity metric could be useful in a large variety of visualization applications. To exemplify one of these applications and better demonstrate the properties of the learned metric, we use it to perform search for occurrences of similar behavior. We developed a prototype system that enables the user to do this search interactively to explore an ensemble dataset (Sec. 5).

Our prototype's main feature is the example-based search over spatiotemporal patches, i.e., 2D+T rectangular subsets of the data. The user can select a few patches containing behavior of interest, and the system returns others containing similar behavior. The similarity is determined using our metric, encoding the query patches and measuring distances to other patches in the learned feature space. While the prototype has only basic features, it showcases our learned metric and its possible applications.

4 Siamese Networks for Similarity Queries

In this section we describe our siamese network (Sec. 4.1), its training procedure (Sec. 4.2) and the resulting similarity metric (Sec. 4.3).

4.1 Pretext task and model

The key component of our approach is the model that we train on the pretext task. As indicated above in Sec. 3, the pretext task is to predict whether two spatiotemporal patches originate from nearby locations in space and time. As our model, we use a neural network that follows the siamese architecture. A typical siamese network takes two inputs (in our case, 2D+T patches), passes them through an encoder and computes the distance between the encodings. Crucially, both inputs use the same encoder, i.e., they share weights. This ensures that the inputs are projected into the same space in which distances will be computed.

In our search application, we will be computing distances to several patches provided by the user, not just one. To improve the performance in this regard, we add a modification to the siamese model. Specifically, we replace one of the input patches with a set of k patches, here called *the query*. During the training, these patches are sampled from the same spatiotemporal neighborhood, and the network's task is to predict whether they also share the neighborhood with the other input patch, called *the test patch*.

An illustration of the model architecture is presented in Fig. 1. Each input patch (the query patches and the test patch) goes through the encoder, which is a convolutional network. All invocations of the encoder use the same shared weights. Next, we compute an element-wise distance between each of the query patch encodings and the test patch encoding. These distances are then averaged (again, element-wise) and used as input for a fully-connected layer that acts as a simple classifier predicting whether the inputs came from the same neighborhood. We use element-wise L1-distance following [42]. The element-wise aspect is particularly important in our self-supervised case: if instead we would have a vector distance producing just a scalar value, the pretext task will have to be solved in the encoder already, which would specialize the learned representation to the pretext task and make it less general. Thus, we use the element-wise distance to let more information pass through the distance operator, shifting the pretext task decision towards the classifier.

Layer	Units/Ch.	Activation	Kernel size	Strides
Conv3D	64	ReLU	(1, 3, 3)	(1, 2, 2)
Conv3D	128	ReLU	(1, 3, 3)	(1, 1, 1)
Conv3D	128	ReLU	(3, 3, 3)	(2, 2, 2)
Conv3D	256	ReLU	(1, 3, 3)	(1, 2, 2)
Reshape				
FC	256	ReLU		
FC	256	Sigmoid		

TABLE 1: The architecture of our convolutional encoder, top-to-bottom. We use a sequence of strided convolutions followed up by a few fully-connected layers to encode each input patch.

4.2 Model training

The model is trained on data sampled from an ensemble of spatiotemporal volumes. Each data point consists of several patches: the test patch and the set of k query patches. Half of the points represent the positive class, i.e., query and test patches coming from the same neighborhood, and half represent the opposite class. In either case, we first determine the location of the test patch by uniformly sampling a random ensemble member and then uniformly sampling a spatiotemporal location within it. This makes sure that ensemble members of different sizes are equally represented in the training data. For points of the positive class, the query patches come from the same member as the test patch, sampled uniformly from a spatiotemporal neighborhood around the test patch. The neighborhood is defined by the maximum spatial and temporal offsets o_s, o_t. This means that all query patches are at most o_s cells and o_t timesteps away from the test patch. For negative points, we first randomly choose an anchor location that is not in the neighborhood of the test patch. We then uniformly sample k patches around the anchor location with maximum offsets o_s, o_t, like the positive case but around a different location. Thus, the query patches always represent similar behavior, which makes them more suitable for distance averaging and improves our latent space. When choosing the offset parameters, we generally try to make offsets larger to make the pretext task more difficult while still training to high accuracy (in appendix Fig. A1 we also show that the method is robust to the choice of parameters).

After the data is collected, we train the model using standard ML practices. We use 20% of the data as a hold-out validation set to monitor the model's generalization performance and perform early stopping. Overall we observed that higher accuracy and lower overfitting on the pretext task leads to better performance of the derived similarity metric. Implementation details of the model and its training follow in Sec. 5.

4.3 Similarity metric

Once the model is trained, we use the encoder to compute patch similarity. With encoder f_e and two spatiotemporal patches p_1, p_2, our patch similarity metric d (technically, dissimilarity) is determined by the L1-distance between the encoded patches:

$$d(p_1, p_2) = \|f_e(p_1) - f_e(p_2)\|_1 . \tag{1}$$

As described in Sec. 3, we demonstrate the utility of this metric by performing interactive queries. To provide results for user queries, we use our similarity metric to construct a ranking score. A query Q consists of two sets of spatiotemporal patches: a set of positive examples Q_+ containing behavior the user is interested in; and an optional set of negative examples Q_- containing behavior that the user would like to exclude. The negative examples can help the user narrow down the query and filter out false-positive results. We define the ranking score r of some patch p to be the average similarity metric between the patch and the query, where the sum of distances of negative patches is subtracted from the sum of distances of positive patches:

$$r(Q_+, Q_-, p) = \frac{1}{\|Q_+\| + \|Q_-\|} \left(\sum_{p_+ \in Q_+} d(p, p_+) - \sum_{p_- \in Q_-} d(p, p_-) \right) \tag{2}$$

The ranking score is therefore low for patches that are similar to the positive part of the query and dissimilar to the negative, indicating a good match. The score is normalized to make the values more comparable across queries of different sizes. Note that during the model training we used only a single set of patches but sampled both positive and negative examples to learn a good similarity metric. This implicitly supports our ranking score that has the positive and the negative parts.

5 Implementation & Prototype System

Network implementation. The architecture of our convolutional encoder is depicted in Tab. 1. Since the ensemble datasets in this paper are all 2D+Time, we use 3D convolutional layers to perform both spatial and temporal convolutions, taking an input with dimensions $batch, time, height, width, channels$. For the 3D+T implementation in Sec. 7.3 we replace all the 3D convolutional layers (2D+T) of the encoder with 4D convolutions (3D+T), using the same strides and sizes. The 4D convolution takes a tensor with dimensions $batch, time, depth, height, width, channels$, and is implemented as a series of 3D convolutions along the time dimension. The output of the final fully-connected layer is used as the encoding of the input. After applying the encoder to each patch, we compute element-wise L1 distance between each query patch and the test patches (see Fig. 1). The distances are then averaged componentwise. Finally, the average distance vector is passed to the classifier. The classifier is a single fully-connected layer with one unit that has the sigmoid activation. During the training, we set the number of query patches $k = 2$ for all our models. Increasing the k further did not harm the final performance but also did not seem to improve it, so we chose the lower value.

The process of sampling training data is described in Sec. 4.2. For all datasets, we downsample spatially by the factor of two and then sample $500,000$ data points (50,000 for the non-ensemble Isabel dataset), each consisting of $k+1$ input patches. We split the points 50:50 between the positive and negative classes, but other ratios could be explored. We also perform data augmentation: patches are randomly mirrored along the spatial axes, and their values are scaled with a random coefficient $\in [0.5, 1.5]$. We train the model using the Adam optimizer [43] with a learning rate of 10^{-4} and a batch size of 32. As a loss function we use binary cross-entropy and an L2-regularization term with a lambda of 2×10^{-4}. During training, we monitor the loss on the held-out validation set and terminate the training when the validation loss stops improving.

To perform search using the ranking score from Sec. 4.3, we first need to obtain the encodings of all the candidate patches. We can choose how densely to sample the candidates, trading off matching time for spatial precision. We extract patches that are non-overlapping in time. When using patches that do not cover the whole spatial domain (Sec. 6), the patches are extracted with spatial stride equal to the quarter of their size. This introduces some

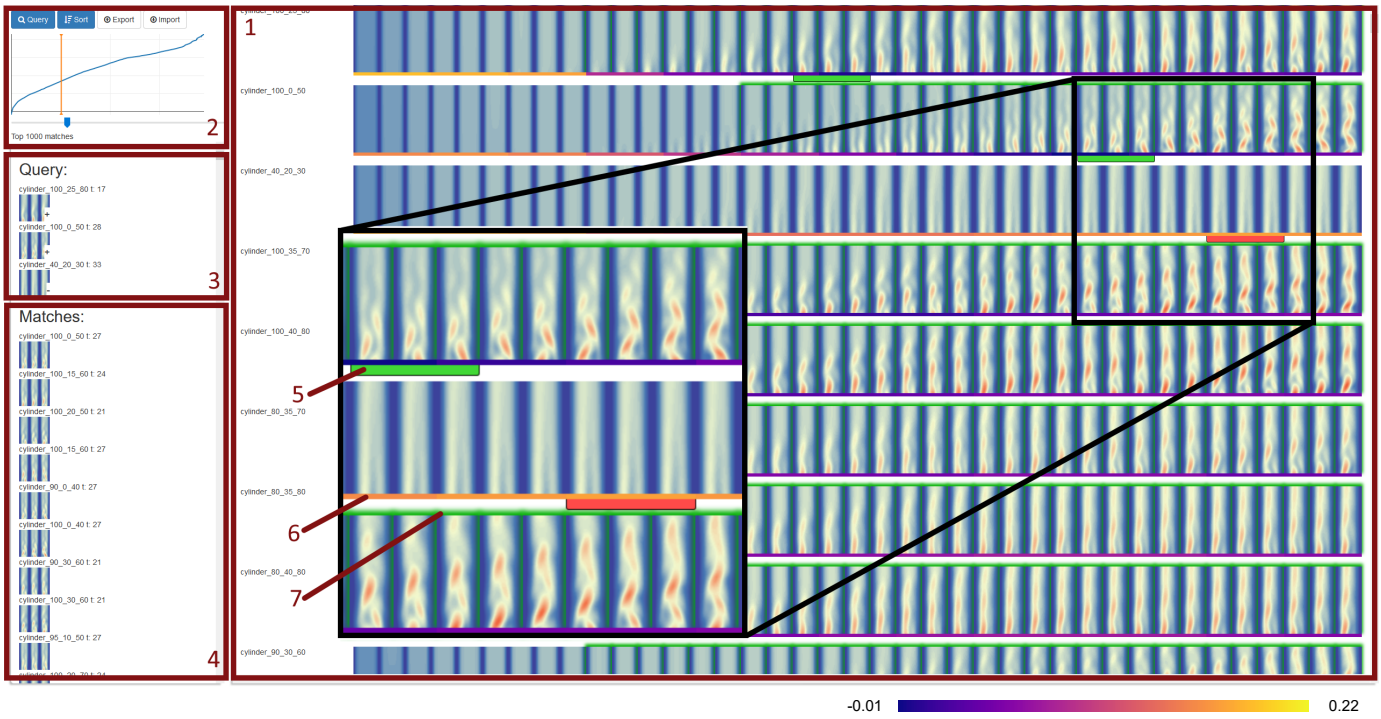


Fig. 2: Our interactive prototype for navigating an ensemble of spatiotemporal data, demonstrating query results on the cylinder dataset. In the navigation panel (**1**) we render a timestep rollout of all the members, showing markers for the query patches (**5**), ranking score (**6**, colorbar below) and highlighting the timesteps that match (**7**). In panel **2** we show sorted ranking scores of all patches in the ensemble. This gives an impression of their distribution and allows us to select which patches are considered to be a match. In panels **3** and **4** we show the contents of the query and the full list of matching patches.

redundancy, since the patches overlap, but enables us to better study the spatial properties of our method.

Prototype system. To better demonstrate the utility of our ranking score and learned similarity metrics in general, we developed a prototype system for navigating ensembles of spatiotemporal data. This is a challenging task, especially for ensembles with a large number of members, making it prohibitive to manually search for the behavior of interest. Our system addresses the problem by allowing the user to find instances of the behavior with only a handful of examples. This reduces the effort for the user, as well as avoids rendering large amounts of data at once. It also reduces the risk of missing good matches compared to the manual search.

In our prototype, we limit ourselves to working with patches that spatially cover the whole domain, i.e., temporal slices (we investigate spatial aspects in Sec. 6.1). The front-end interface of the system is shown in Fig. 2, with the cylinder dataset loaded (depicting flow around a cylinder at the bottom of the domain). The navigation panel (1) is the main component of the UI. Here we present each ensemble member as a row containing a list of renderings of its timesteps. The user can click on a timestep to include a patch starting on this timestep into the query. We show the contents of the query using green and red marks (5) in the navigation panel, as well as a separate list in the query panel (3). The green marks correspond to positive examples and red to negative. Once the query is formed, the user can use the controls (2) to execute the query, loading the results from the backend. The results are presented in several ways. First, we encode the ranking score (similarity) of every patch in the ensemble as a colored bar right below the timestep renderings (6, colorbar below). The best N matching patches are highlighted with green (7) in the navigation panel and are also shown as a list in the matches panel (4). To allow the user to configure the value N, we show the ranking score graph (2), plotting the sorted ranking scores of all the patches in the dataset. This gives an overall impression of the distribution of ranking scores and enables the selection of the cut-off line, i.e., how many top matches to display. Finally, the user can also sort the members in the navigation panel, according to the current query: members mentioned in the query itself are shown on top, followed by the members with the most matches. This makes it possible to quickly see the most relevant members for the current query and its results.

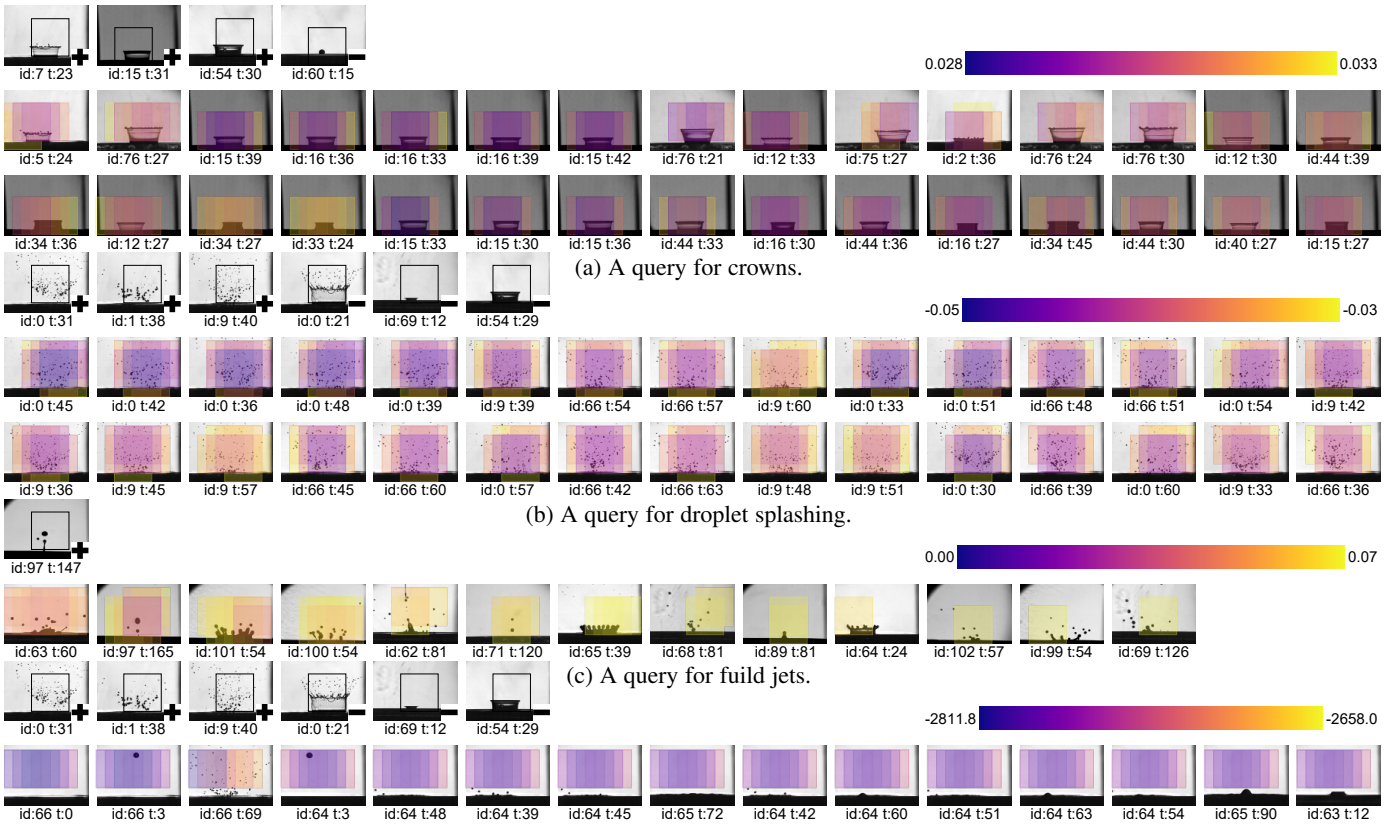
6 Qualitative Evaluation

In this section, we perform a qualitative evaluation, demonstrating specific queries and their outcome (Sec. 6.1). We then show how our approach compares to results manually crafted by a domain expert (Sec. 6.2) and discuss her feedback (Sec. 6.3).

6.1 Query results

First, we demonstrate the search results on the "droplet splash" dataset, which is an ensemble of experiments containing monochrome camera images of a single droplet impacting a thin liquid layer. Using different fluids, droplet size and impact velocity result in different impact regimes such as deposition, bubble formation or splashing, i.e., secondary droplets separating. There are 110 members, each ranging from 44 to 538 timesteps with a spatial size of 224×160. We used patches with a temporal size of three timesteps and a spatial extent of 50, and the neighborhood size is defined as $o_t = 9, o_s = 60$. This experimental ensemble is particularly challenging, as it contains images with differences in panning, zoom and illumination. Conventional similarity metrics

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3101418, IEEE Transactions on Visualization and Computer Graphics



(d) MSE results for the query from **b**.

Fig. 3: Query results on the droplet splash dataset. In the top left, we render the query patches (their first timestep). Positive examples are marked with **+** and negative with **-**. We take 500 best matching patches and render the timestep they start on, sorted by the number of matches in that timestep (to see the frames with many good matches first). The member ID and the timestep index are printed below each frame. We color the matching patches based on their score, where blue means better matches. As we see in **a, b**, our method finds many diverse examples of the queried behavior. Even with a single patch of rare behavior (**c**) we can find its other instances (here we show only one frame per member for brevity). Compared to the MSE baseline in **d**, our method returns results that are much more useful. An extended version of this figure can be found in the appendix.

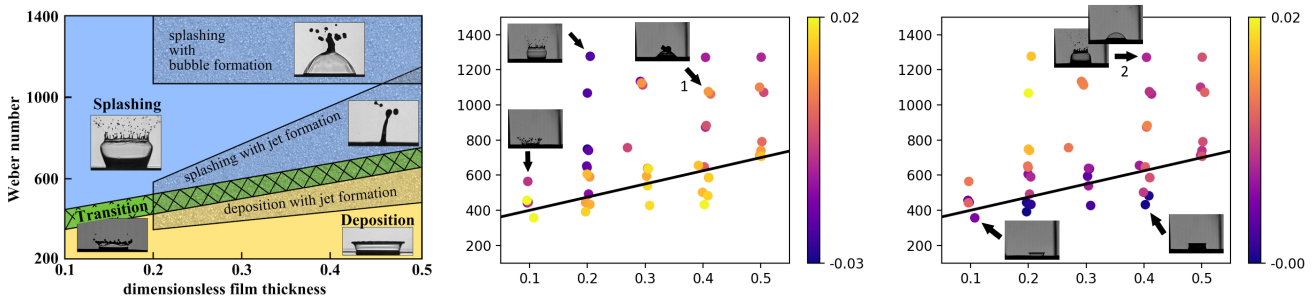
often yield poor results under these conditions (see discussion below). However, since our model is trained on the dataset, it is able to learn some of these invariances and performs robustly.

In Fig. 3 we present the results of different queries. First, we performed a query searching for fluid crowns (Fig. 3a). The query contains three positive examples of crowns and one negative example with the initial droplet collision that occurs just before a crown is typically formed. In the results, we present the best matching patches rendered in the context of the data. Specifically, we took 500 best matching patches and grouped them by the timestep they begin on. We render timesteps that have the highest count of matches, highlighting the matching patch locations. The patch rectangles are colored according to their ranking score and rendered in worst-to-best order. Thus, in the locations where multiple patches overlap, we see the best-ranking score for that location. We observe that the results contain examples of crowns with varying shapes coming from many different members (specified by their ID). Despite the significant optical differences between the experiments, the method is able to detect relevant behavior. Furthermore, we see that beyond determining the member and the timestep where the behavior took place, we also receive spatial information: the best matching patches are well aligned spatially with the feature of interest. For an example of constructing queries interactively, see the supplementary video.

Another query is shown in Fig. 3b. Here we search for instances of splashing, i.e., secondary droplets forming after the collision. We see that the results again contain a diverse set of patches matching our query. Even though direct differences between the patches with secondary droplets are large, our method can still find them. We found a few examples that initially look like crowns (see also appendix), but on closer inspection, they turn out to be crowns that have just transformed into a spray of secondary droplets, thus matching the query. This hints at a property that follows from the training process: since we train the model by encouraging spatiotemporal coherence, we expect the matching to be somewhat "fuzzy", also having low distances to spatially and temporally neighboring behavior.

For comparison, in Fig. 3d we show the same query but using MSE as the similarity metric for the ranking score. The search does not yield meaningful results, among others returning patches with empty space, since MSE heavily prioritizes the background and misses the droplets. It returns one valid timestep by matching the background, but this is only a rare outlier. In contrast, the top model-based results (Fig. 3b) contain precisely the relevant patches with splashing droplets.

Although crowns and secondary droplets are the key features of this dataset, we also want to investigate if the method is capable of finding rare events. We constructed a query with just a single



(a) Results by the domain expert [44]. (b) A query for splashing. (c) A query for deposition.

Fig. 4: Comparison of manual domain-specific results to our method. **a:** Parameter space map of droplet impact regimes manually constructed by the domain expert. **b:** Ranking scores of ensemble members for a splashing query (see Fig. 3a). We find structure similar to (a), with best matching experiments (in blue) being located in the top-left quadrant of the parameter space and a similar transition region (the transition line from (a) plotted for comparison). We also found a poor-matching outlier region (marked "1"). A closer investigation led to the discovery of a region containing bubble formation and deposition that was previously unknown to the expert. **c:** Ranking scores for a deposition query. We find best-matching experiments to have lower Weber number or higher film thickness, in accordance with domain-expert results. A few experiments (marked "2") are matched moderately well by both queries because they display both bubble formation and splashing.

patch containing a fluid column with a droplet separation (Fig. 3c). This figure, shows only the best frame of each matched member to conserve space (full result in the appendix). Here we note two types of matches. First, we see some crowns that have distinct fluid columns with droplets separating (id 63 and 101). Since separation occurs simultaneously in several locations, we get a lot of matches, and the corresponding timesteps get sorted to the top. Then, we also see some matches with a single fluid column. Some are from the query member (id 97), but some point towards other members with similar behavior (id 62, 71 and 89). Despite this event being much less typical for the ensemble, our approach was able to find similar instances with just a single example.

Next, we evaluate the temporal aspects of our metric on the "cylinder" dataset. This dataset is a CFD simulation ensemble of flow around a cylinder obstacle. Depending on the Reynolds number and the obstacle configuration, one observes different degrees of turbulence in the flow. We cropped the spatial domain, such that the obstacle is no longer visible (only the channel), so that the matching algorithm cannot "cheat" by comparing the obstacles. We used 300 ensemble members, where each member is a scalar velocity magnitude field with 39 timesteps and the spatial domain size 84×220. For this dataset, we focus on behavior extending over the whole spatial domain, so we use patches with the temporal size of three and spatial size equal to the data domain size, while the neighborhood size is $o_t = 9$. In Fig. 2, we used our prototype to perform a query for turbulent members, providing two examples of turbulence and one negative example containing laminar behavior.

As we can see, after sorting the members by the number of matching patches, we find many other members containing turbulent behavior. Notice that the exact geometry of the flow can be very different, but the model still considers them similar due to invariances learned during training. We also observe that the matching is successful temporally: we identify the point in time when the turbulence starts to occur.

6.2 Parameter space analysis

Next, we evaluate our results on the "droplet splash" dataset in comparison with the extensive manual analysis by a domain expert. This dataset was collected to study droplet impact regimes wrt. experimental parameters such as fluid viscosity, droplet velocity, film thickness, etc. In the previous analysis [44], the expert has taken a subset of the ensemble depicting a particular fluid and has manually constructed a regime map of the parameter space, shown in Fig. 4a. As we change the droplet velocity (Weber number) and the film thickness, we observe qualitatively different outcomes, with a thicker film and a slower droplet leading to cleaner deposition.

To compare our method's ability to detect different impact regimes, we performed two queries, one for splashing (also in Fig. 3b) and another one for deposition. Then, we computed a ranking score for each ensemble member by simply taking the minimum score of all the patches from a given member. Thus, if a member contains a well-matching patch, the member itself is considered to be well-matching. We visualize the member scores positioned in the parameter space to compare them to the regime map that was constructed manually by the domain expert.

In Fig. 4b we show the splashing query results. The best-matching members are located in the top-left corner (high velocity, thin film), and we can see that the score degrades as we move bottom-right, forming a transition region, which aligns well with the expert results. We plot the transition line from the expert map to make the comparison easier. Here we noticed a region with some poorly-matching outliers in the top-right quadrant (marked with "1"). After consulting with the domain expert and checking the experiment images, we found out that there is a bubble deposition subregime, which explained why the splashing query was not matching well. More importantly, its existence was previously unknown to the expert, highlighting one of the strengths of our method: being able to detect features of the data that the domain expert might have overlooked.

In Fig. 4c we show results of a deposition query. Here, as expected, we get an inverse result: best-matching members contain clean deposition and are located in the bottom right corner of the parameter space (low velocity, thick film), and the splashing members (top-left corner) have high scores, again corresponding well to the results by the expert. We found one interesting outlier (marked with "2") that is matched reasonably well by both the splashing and the deposition queries. Upon inspection, we found that it contains a splashing phase, followed by a bubble

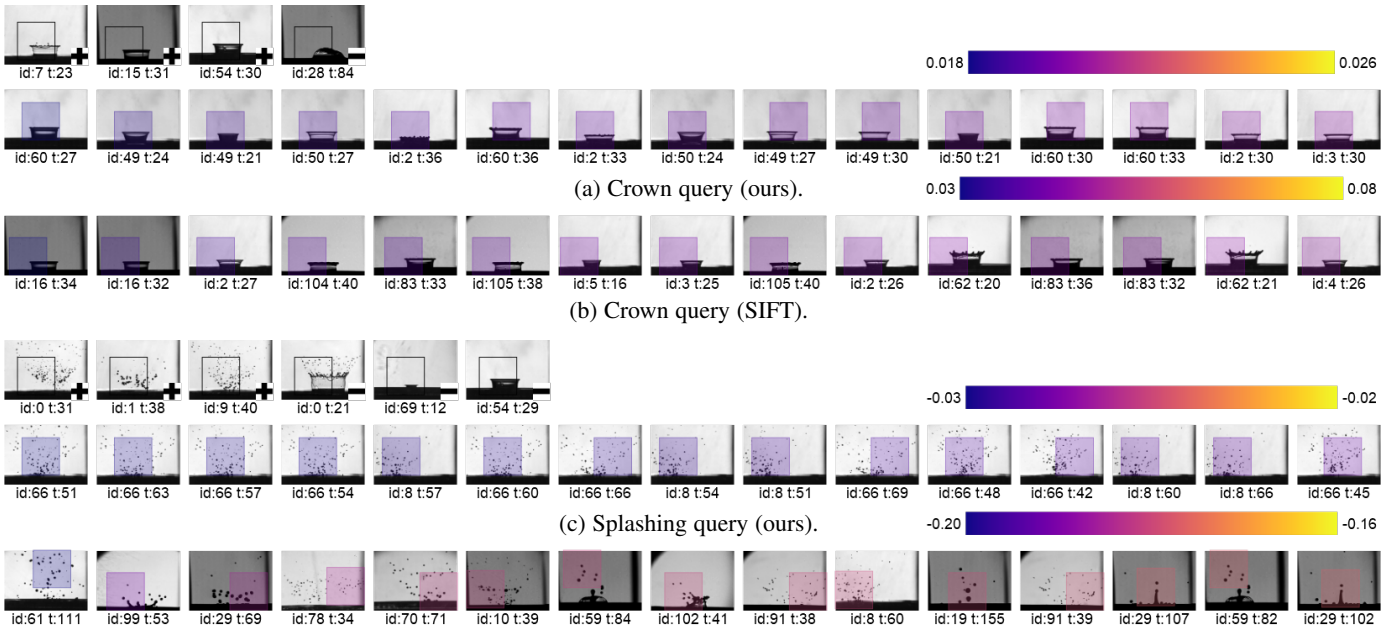


Fig. 5: Comparison of our similarity metric to SIFT on the droplet ensemble. **a, b:** we see that both methods produce good results on the fluid crown query, as it is very suitable for the SIFT descriptors. **c, d:** However, on the splashing query our method returns more robuts results, as SIFT is struggling to match small disperse droplets. An extended version of this figure can be found in the appendix.

phase, and since we use the minimum function to aggregate the patch scores for this figure, the member was matched well by both of the queries, each matching an appropriate time range.

Overall, we confirmed that our technique yields meaningful results when applied to data from this domain, successfully finding different types of behavior with a few simple queries. The results exhibit a high degree of similarity to the manually-constructed regime map, which the domain expert also pointed out (Sec. 6.3).

6.3 Domain expert feedback

As part of our evaluation, we discussed our results on the droplet splash data with a domain expert in droplet dynamics. She collected and extensively analyzed the experimental ensemble data. We performed searches for several droplet regimes (e.g. Fig. 3), which were previously studied by manually selecting relevant members and time ranges that were fed into ad-hoc Matlab scripts for further analysis. The expert noted very good agreement for detection of droplet splashing and deposition with bubble formation and fair results for queries of crown-forming deposition and jet formation. However, the query for splashing with bubble formation had a lot of unexpected matches. Here our method produced many matches containing crown-forming splashing, confusing bubbles with transparent crowns. Our parameter space comparison (Sec. 6.2) was viewed very positively: “In my opinion, the results are very good. The splashing limit is quite well reproduced in your maps.”. The expert also pointed out that we found a small region that was previously inaccurately classified: “You detected a small region of bubble formation and deposition, which we did not recognize or to be precise which we counted as splashing”.

She was very optimistic about the utility of the system in her workflow: “I think such a tool would have been and would be very useful for us because all the regime maps and splashing limits were derived by manually digging through every video and deciding what we see. [...] A tool like yours would make it much easier if we are looking for example at a special feature like jet formation.” As a suggestion, she noted that the current results for splashing with a bubble could be further improved by performing iterative searches, i.e., first searching for a generic regime and then narrowing down the results with more specialized queries. Overall, we received very positive feedback about our technique, suggesting that our domain-agnostic method can act as a useful building block in addressing domain-specific problems.

7 Comparative Evaluation

In this paper, we aim to demonstrate that machine learning can be used to construct an effective spatiotemporal similarity metric in a domain-agnostic fashion. This section discusses existing approaches to computing a spatiotemporal similarity metric and perform a qualitative comparison to some of these techniques. In the next section (Sec. 8), we also compare quantitatively to many more alternative methods.

7.1 Alternative approaches

A similarity metric is an important component of many ensemble visualization approaches. Thus many other possibilities have been proposed. These can be roughly split into two categories. First, there are mostly generic vector, image or distribution distances, such as MSE (L2), L1, EMD, SSIM and local histogram differences. Such similarity metrics can be applied directly to the raw data and are a very common choice in modern ensemble visualization literature (see Table 2 in the survey by Wang et al. [1]). The other group of methods extracts domain- or problem-specific features from the data (e.g., vorticity or λ_2 for CFD datasets) and computes distances in that feature space.

The former metrics are the most appropriate to compare with our technique, as they are widely used in recent work and are similarly domain-agnostic. We compare our method to many

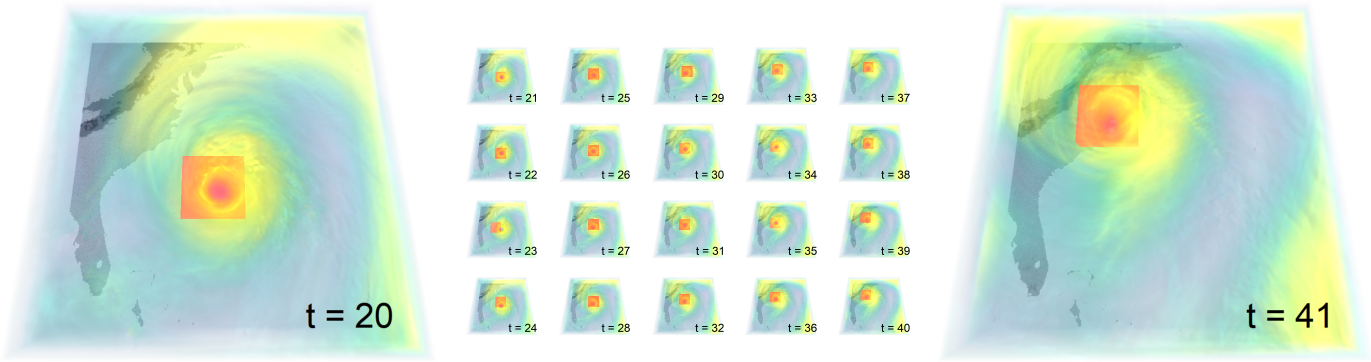


Fig. 6: Comparison of our method to Wang et al. [13] on the example of the 3D Isabel dataset. The figure is analogous to Fig. 1 in their paper. We provided a single query patch of the hurricane eye in timestep 20 (left) and then render the best matching patch in the following timesteps. As we see, similarly to the method of Wang et al., we are able to track the eye of the hurricane.

of these metrics in our quantitative evaluation in Sec. 8. The comparison to the latter group is less appropriate since we are proposing a general domain-agnostic technique. While problem-specific solutions might outperform a general method, they cannot be applied to data from different domains, which is the core motivation for a learning-based similarity. Nevertheless, we still compare with two specialized methods in Sec. 7.2 and Sec. 7.3. We have chosen these approaches because they rely on a commonly used algorithm (SIFT), are somewhat data-agnostic (within their domain) and do not rely solely on basic distance metrics, improving the diversity of our evaluation. With this, we demonstrate that our generic metric yields similar performance while not relying on domain-specific features.

7.2 Comparison to SIFT

In this section, we compare the results of our similarity metrics to those computed with SIFT [45]. SIFT is a computer vision technique for computing image keypoints and their local feature descriptors that are invariant to illumination, scale and orientation changes. Since SIFT is a robust matching algorithm that can, in principle, be applied to any scalar field, it provides an interesting comparison for our approach.

We applied the SIFT algorithm (which computes sparse image correspondences) to perform our patch-based queries as follows. First, we compute SIFT keypoints for all the timesteps and all the members of the ensemble dataset. By computing the keypoints over the whole spatial domain (as opposed to a given patch), we make sure that any large-scale keypoints would still be correctly extracted. Next, we define the SIFT distance between two patches as the minimum SIFT-descriptor distance between any keypoints lying within the patches. Given a query consisting of several positive and negative examples, we compute the ranking score according to Eq. 2, i.e., the average SIFT distance from all the query patches to a candidate patch, where negative patches contribute negatively. This way, we compute the SIFT-based ranking score from the query to every dataset patch and find the best matches.

The results for the "droplet splash" are presented in Fig. 5. In Fig. 5a and Fig. 5b we show the results for the same crown query obtained using our model and the SIFT-based method. As we can see, SIFT produces a very accurate matching, which is not surprising: the dataset of camera images and the characteristic corners of the crowns provide an ideal application scenario for SIFT. Our model also successfully finds crowns in the data, though it also sometimes matches similar bubble deposition (this is discussed in more detail in Sec. 6.3).

Next, we perform a query for splashing (Fig. 5c, Fig. 5d), and here we see that SIFT performs much less robustly, also returning some crown deposition cases. The reason behind this is that a spray of small droplets does not yield robust SIFT keypoints, thus leading to worse performance. In contrast, our method has learned dataset-specific features and is thus able to return accurate matches.

Overall, we found that our method performs well compared to SIFT-based matching on this dataset of camera images and even outperforms it in some scenarios. Furthermore, we were unable to apply SIFT to our "cylinder" CFD ensemble (described in Sec. 6.1) because SIFT is not extracting any keypoints for most of the timesteps of this smooth dataset, while our method still performs robustly. This once again demonstrates the advantages of our generic similarity metric.

7.3 Comparison to Wang et al. [13]

In order to demonstrate that our approach can qualitatively match the results obtained with more traditional non-ML methods, we compare to the results from Wang et al. [13]. They introduced a method for finding similar regions in vector fields, which is used to return matches for a user-provided query region. First, they extract a pre-defined set of vector field features (called traits): divergence, the norm of Jacobian, etc. Then, SIFT-matches are computed in each field and used as match candidates. This serves to reduce the search space, as well as align the candidate regions with the query. Finally, they use the weighted L2 norm to evaluate the similarity of candidates to the query region. Notably, the method relies on a fixed set of features and can only be applied to vector fields. As discussed in Sec. 7.1, our approach prioritizes generality and thus only relies on a single raw scalar field. We also compare with applying SIFT to raw data in Sec. 7.2.

To compare to the results by Wang et al. we apply our method to the Isabel dataset, using the 3D+T version of our encoder (Sec. 5) and a 3D+T patch size $3 \times 15 \times 50 \times 50$, matching the proportions of the Isabel dataset. Unlike Wang et al., we are only using a single scalar field as input – the velocity magnitude. After training the model, we performed a query that aims to reproduce the result in their teaser image (Fig. 1 in [13]). In the query, we provided a single patch near the eye of the hurricane at timestep 20, and in Fig. 6 we visualize the best-matching patch in each of the following timesteps, rendering it as a transparent red box. Here

		Ours				VGG				MSE				EMD				Hist-EMD			
Feature	Patches	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100	C	P10	P50	P100
turb	1+0-	**72.9**	**100.0**	84.0	**77.0**	65.0	**100.0**	**90.0**	72.0	42.4	90.0	50.0	45.0	38.5	**100.0**	62.0	39.0	56.5	70.0	56.0	54.0
turb	2+0-	**67.6**	**100.0**	88.0	**74.0**	64.1	**100.0**	**90.0**	73.0	39.4	90.0	50.0	42.0	38.5	90.0	60.0	41.0	60.0	90.0	48.0	54.0
turb	3+0-	**78.2**	**100.0**	**96.0**	**88.0**	66.8	**100.0**	94.0	77.0	36.8	70.0	44.0	41.0	35.0	80.0	42.0	36.0	59.1	20.0	52.0	56.0
turb	1+1-	**80.3**	**100.0**	**100.0**	**83.0**	69.4	**100.0**	94.0	75.0	53.8	90.0	56.0	54.0	52.1	50.0	64.0	54.0	54.7	60.0	56.0	56.0
turb	2+2-	**82.6**	**100.0**	**96.0**	**87.0**	69.4	**100.0**	92.0	76.0	52.1	90.0	56.0	52.0	47.6	50.0	64.0	53.0	57.4	50.0	54.0	56.0
turb	3+3-	**88.5**	**100.0**	**100.0**	**92.0**	76.5	**100.0**	**100.0**	81.0	52.1	70.0	58.0	53.0	48.5	50.0	64.0	53.0	56.5	60.0	54.0	56.0
lam	1+0-	63.7	**100.0**	**100.0**	**90.0**	**73.7**	**100.0**	94.0	89.0	64.2	**100.0**	76.0	77.0	65.1	**100.0**	94.0	86.0	60.9	90.0	54.0	49.0
lam	2+0-	66.2	**100.0**	**100.0**	**93.0**	**73.7**	**100.0**	**100.0**	90.0	64.2	**100.0**	**100.0**	87.0	64.2	**100.0**	**100.0**	92.0	60.9	90.0	54.0	49.0
lam	3+0-	64.6	**100.0**	**100.0**	**95.0**	**73.7**	**100.0**	94.0	90.0	65.1	**100.0**	98.0	86.0	65.1	**100.0**	94.0	91.0	62.8	90.0	56.0	59.0
lam	1+1-	**79.6**	**100.0**	**100.0**	**100.0**	71.7	**100.0**	90.0	88.0	62.3	60.0	72.0	68.0	61.7	50.0	58.0	71.0	63.6	90.0	52.0	50.0
lam	2+2-	**81.2**	**100.0**	**100.0**	**100.0**	76.4	**100.0**	98.0	91.0	64.0	70.0	86.0	75.0	63.1	70.0	66.0	70.0	63.1	90.0	58.0	49.0
lam	3+3-	**83.5**	**100.0**	**100.0**	**100.0**	81.9	**100.0**	98.0	93.0	68.3	**100.0**	90.0	77.0	64.9	60.0	72.0	73.0	79.8	**100.0**	92.0	94.0

TABLE 2: Results of manually constructed queries on the cylinder flow ensemble. Each row depicts metric scores for one query. A query contains either patches with turbulent (turb) or laminar (lam) behavior. The 'Patches' column specifies how many positive and negative patches each query contains. Different queries are constructed by editing previous ones. For example, the turbulent query 'turb 3+3-' contains the same three positive patches as 'turb 3+0-', but we added three negative non-turbulent patches. We show four quality metrics: coverage (C) and precision (P) at three different ranks. We render query matches in Fig. 7. Our model achieves significantly better results than the various baselines.

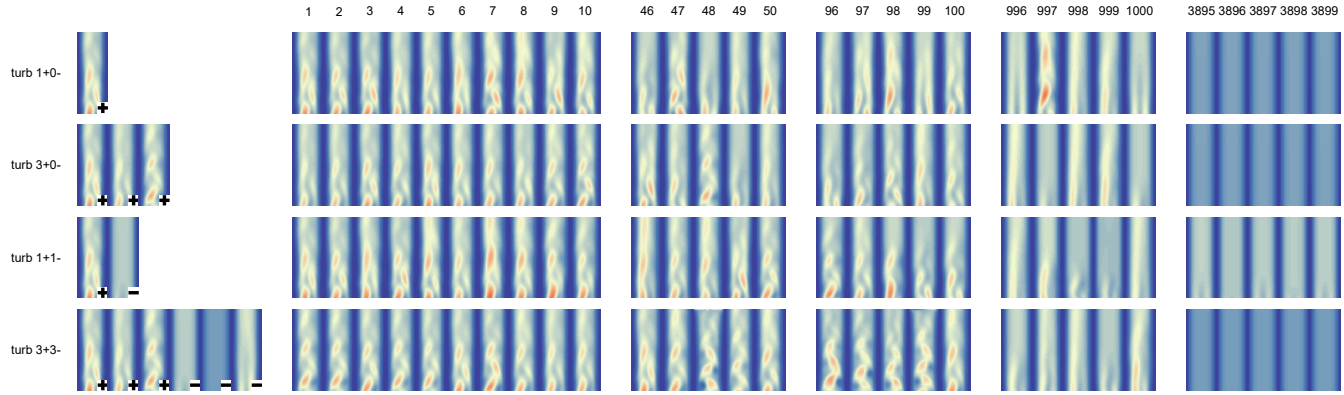


Fig. 7: Corresponding renderings to matches of turbulent queries from Tab. 2. Each row corresponds to a query. On the left are the query patches, and on the right, we show respective matches. We only render the first frame of each patch. We show the top 10 matches as well as some examples of lower matches. The icon on the query patches indicates whether a patch is a positive (+) or a negative (-) example.

we sampled candidate patches with stride equal to a quarter of the patch size to provide better spatial precision. As we can see, we are able to reproduce the result, effectively tracking the eye of the hurricane, while using only a single scalar field and not relying on any vector field features. This demonstrates the model's ability to learn relevant features during the self-supervised training.

8 Quantitative Evaluation

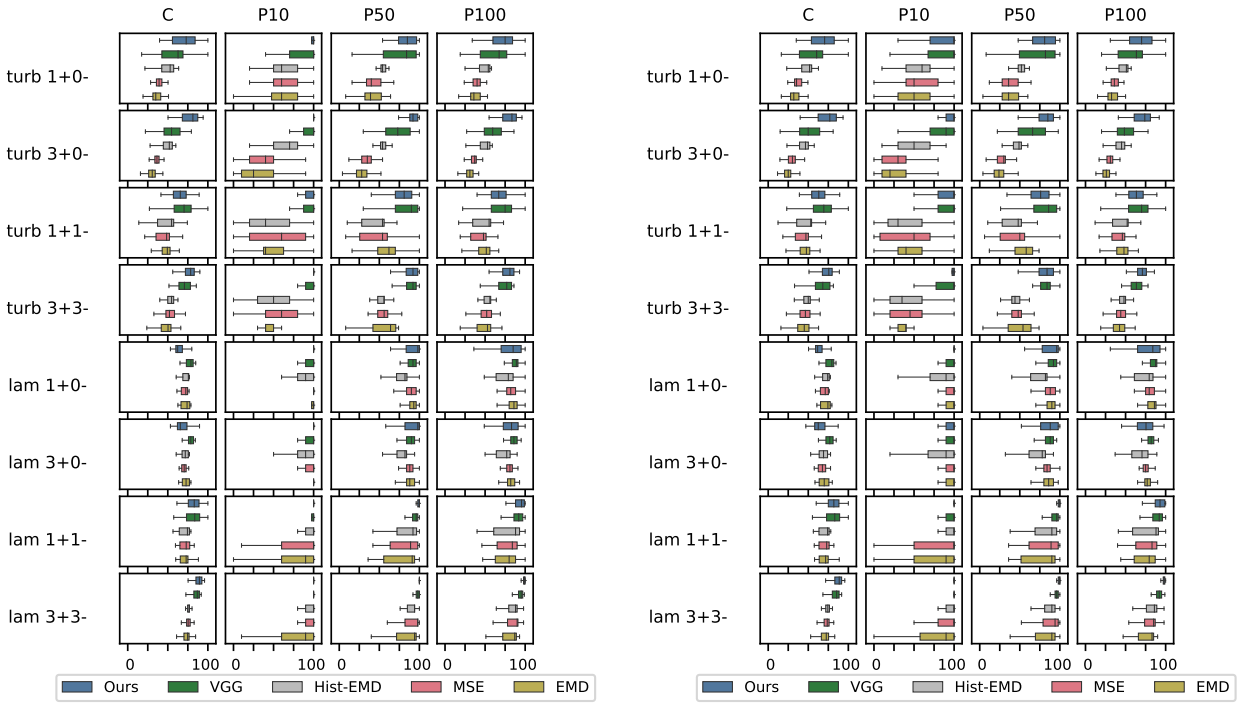
Next, we perform a quantitative evaluation of our method, aiming to assess its accuracy and compare it to alternative approaches. However, we do not have the ground truth describing what behavior occurs in different parts of the data or the similarity of different data patches. In fact, this is the very problem that we are addressing with our method. Thus, to enable the evaluation, we have manually labeled a small subset of the "cylinder" simulation ensemble (25 out of 300 members). This ensemble exhibits a range of different outcomes, however, there are two overall behavior types that we can approximately distinguish: turbulent and laminar flow. Importantly, this behavior occurs across the whole spatial domain, which means that we only need to label timesteps.

With this, we assess accuracy using different search quality metrics (Sec. 8.1), both for our method and four alternative approaches (Sec. 8.2). Then in the following sections, we discuss results on manual as well as random queries, quantify model variance and its generalization performance.

8.1 Quality Metrics

Given a query, our method assigns a score to each patch in the data, effectively ranking the patches based on how well they match the query. Assuming that the positive patches in the query all contain some sought-after behavior A and all negative patches do not contain behavior A, we get the binary ground truth of patch relevance: patches with behavior A are relevant, and patches without behavior A are irrelevant. With this, we compute precision at three different cut-off ranks of 10, 50 and 100, e.g., Precision@50 is the percentage of relevant patches in the top 50 matches. In an interactive search scenario, it would be unreasonable to consider cut-off values above a few hundred for user investigation.

Aiming to also measure the result completeness, we designed a metric suitable to our problem. We compute timestep *coverage*, i.e., what percentage of timesteps exhibiting the queried behavior is included (covered) in the top N patches. And we determine N to be the minimum amount of patches needed to cover all the relevant timesteps. A ranking score that prioritizes patches with matching behavior will have higher coverage. The metric is less sensitive to the changes in the patch size, whereas precision may be more optimistic when the patch size is small and many candidates are available. Furthermore, the coverage metric considers how patches are positioned near the borders of feature regions: missing a patch that barely touches a relevant region has less penalty than missing a patch in the middle of the region.



(a) Including all results. (b) Excluding query members.

Fig. 8: Measuring the distribution of search quality for "cylinder" wrt. randomly sampled queries. Each row represents a query type, e.g., 'turb 3+1-' means queries containing three random turbulent patches and one random non-turbulent patch. In the columns, we compute the same quality metrics as in Tab. 2, with all metrics ranging from 0 to 100. **a**: Our model shows better results than the baselines, especially on larger queries. We also see that the variance is reduced when more examples are used in the query. **b**: We evaluate the model's generalization by excluding patches from members mentioned in the query. Performance is slightly worse (as expected when removing the best matches), but the model is still successful. This suggests that the model generalizes beyond the pretext task and finds other instances of behavior.

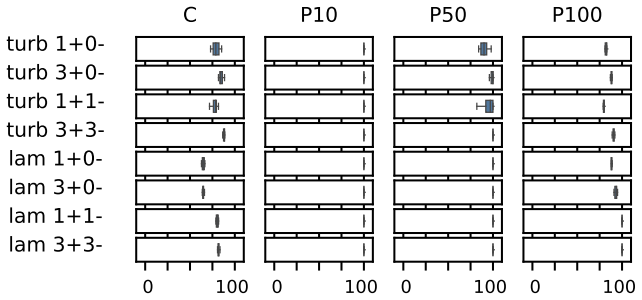


Fig. 9: The variance of search quality wrt. training process. Here we have trained our model ten times and performed the search with each one. We used the same manually constructed queries as in Tab. 2. While some variance is present, it is not significant and decreases with increasing query size.

8.2 Baseline methods

We compare our model against many other similarity metrics: mean squared error (MSE), earth mover's distance (EMD), EMD applied to histograms (Hist-EMD) and a metric based on a pre-trained VGG16 model. In the appendix, we also show results for structural similarity (SSIM), L1 distance applied to histograms (Hist-L1) and fine-tuned VGG variants. To implement search using these metrics, we replace our model-based similarity metric d with the baseline metric when computing the ranking score from Eq. 2. For the MSE metric, we compute the squared difference between the two spatiotemporal patches. We compute the EMD using the POT library [46] implementation of the Sinkorn algorithm [47], computing the distance between the first timesteps, downsampled by the factor of four. Even with these simplifications, it takes around two days to perform the EMD computations for the evaluation on the "cylinder" ensemble. Hist-EMD is computed as the distance between the patch data sorted into a histogram. VGG16 [48] is a large computer vision model (100 million parameters vs. our 2 million) pre-trained on a big image classification dataset. VGG has been shown to generate generic features that are useful for estimating image similarity [49]. By comparing with it, we want to check if our encoder is learning useful problem-specific features or if it can be replaced with a powerful but generic computer vision model. The VGG metric is computed by putting the first frame of each patch through the VGG16 model and computing the distance in its learned feature space. Specifically, we take the output of the "fc1" layer and compute the Euclidean distance.

8.3 Query results

We manually prepared two groups of queries containing typical laminar and turbulent patches. The quantitative metrics are presented in Tab. 2 and the rendered matches are shown in Fig. 7. First of all, we see that overall the model-based search outperforms all the baselines, especially the non-ML methods. Specifically, the model shows both higher precision in the top results and higher coverage. The difference between the methods is generally less pronounced on the laminar queries because this scenario is more favorable for

our baseline methods (especially MSE and EMD), since laminar members often have very small direct differences to each other. We also notice that the other learning-based method VGG achieves better coverage on the first three queries than our approach. This is because our model learns spatiotemporal aspects of the dataset and might, for example, consider slightly turbulent members similar to laminar, which is not accounted for by our binary labeling. Thus, in this particularly simple scenario of laminar behavior, it might perform slightly worse than a computer vision model. However, adding even a single negative patch to the query significantly improves the model's performance, again putting it above all the baselines. Another interesting observation is that the quality of the model's results generally improves as we expand the query. This is not always the case for other methods, e.g., MSE and EMD: especially for the turbulent queries, additional patches sometimes do not improve the result or even worsen the performance. Even a slight temporal shift or a difference in phase can result in very large MSE distances, worsening its results, while the model has learned these invariances during the training. For example, notice in Fig. 7 that some of the matched patches have different phases (velocity peaks do not align). Furthermore, our model is more consistently benefiting from negative query patches, while other methods benefit from them only under certain circumstances.

8.4 Variance quantification

Query sampling variance. Now we aim to evaluate the accuracy across a large number of possible search queries. For this, we defined 12 search query types by specifying sought-after behavior (turbulent or laminar) and the number of patches in the query. Then for each search type, we randomly generated 100 queries, utilizing the labels to ensure that each query contains appropriate patches. We present the results as a table of boxplots in Fig. 8a. Most importantly, we again find that the model performs better than the baselines, and the effects described in Sec. 8.3 are still present when measured over a large sample of queries. However, we also see that non-ML results (MSE, EMD, Hist-EMD) sometimes have lower variance. This is explained by the fact that these methods mostly find nearby patches (due to temporal coherence), which gives consistent albeit poor results. Notice also that our model's results again tend to improve not only in accuracy, but also in terms of uncertainty as more patches are provided in the query.

Model training variance. One potential concern when using ML-models is the variance introduced during the training process, especially when transferring the model's representation to another task. To quantify the impact of model training, including training data sampling, weight initialization and data shuffling, we performed the training ten times and applied the resulting models to the twelve queries from Sec. 8.3. We observe that the model results indeed exhibit some variance, however, it is significantly below the query variance and stays consistently above the baselines (Fig. 9). Additionally, we see that the results are the noisiest when using queries with few patches, but as more patches are added, the uncertainty gets considerably reduced. We believe that this effect is similar to ensemble averaging, as we effectively average several instantiations of our convolutional encoder.

8.5 Model generalization

Another important aspect of a model's performance is its generalization properties across different tasks. Since we train the model on the pretext task, we want to make sure that we "get out" more than we "put in", i.e., that our model does not simply find patches from nearby locations. While finding similar patches from similar locations is useful, ideally the model should also generalize to other instances of similar behavior. To study this, we have used the same setup as in Sec. 8.4, sampling random queries and measuring search result metrics. However, we made a crucial modification: we remove from the list of candidate patches those patches that come from ensemble members mentioned in the query. This way, a model that simply solves the pretext task would only find the members from the query and show poor results on the out-of-query data. The results are presented in Fig. 8b.

Overall we can see that the performance is comparable to our previous experiments in Sec. 8.4, again demonstrating better results than the baselines. This suggests that the model indeed generalizes beyond finding patches from the same ensemble members. We observe some decrease in accuracy, but this is reasonable, since patches from members included in the query usually contain the most similar behavior and are expected to be among the top results. Thus, removing them from the pool of valid matches slightly lowers the quality metrics.

8.6 Performance and ML metrics

On the "droplet splash" dataset, the siamese model was trained with 45GB of data (500k points) on a desktop NVIDIA GTX 2070 graphics card in 4 hours and 20 minutes, achieving 94.7% training and 94.2% validation accuracy in 22 epochs. In our prototype system, the encodings of the candidate patches are precomputed (see Sec. 5), so the query performance is determined by how fast query patch data is encoded and distances to candidates are computed. A single-patch query in our prototype system takes 19ms to execute, where 3ms are spent encoding the patch on the GPU. And a ten-patch query takes 178ms to execute, where again only 3ms are spent encoding the patches (no impact due to parallelism), with distance computations taking up the most time. On the "cylinder", the model was trained in 4h 25min to 97.7%/97.8% accuracy; a one-patch query takes 8ms, and a ten-patch query takes 41ms to execute. On the "Isabel", the model was trained in 3h 15min to 72.0%/69.7% accuracy (if trained further, the model overfits on this smaller dataset); a one-patch query takes 5ms and a ten-patch query takes 31ms to execute. If we sample patches 64 times more densely (Fig. 6), queries take 148ms and 1.3s, respectively. The above query timings are obtained while preloading the ensemble data into memory to remove the IO bottleneck of loading the patch data from disk. When this feature is disabled (or if the ensemble is too large), a naive implementation incurs an overhead of about 100-150 ms per ensemble member present in the query. Note that, in general, we did not heavily optimize our implementation for performance, and we believe there is potential for improvement. In particular, the performance of the prototype system could significantly be reduced with an optimized parallel implementation, a spatial data structure or using precomputed encodings for the query patches as well.

9 Discussion

In this paper, we address an important challenge in visualization: data-driven analysis of large amounts of unlabeled scientific data. While we focus on using our learned metric to perform search, our similarity metric can be useful for various visualization techniques. For example, clustering algorithms are generally based on quantifying distances, which our similarity metric could provide.

Some algorithms, e.g., hierarchical clustering, also use the distance between a cluster and a point to compute clusters progressively. Here, our ranking score can be used to help form more meaningful clusters. Furthermore, a similarity metric can be utilized to perform projections, producing both a more meaningful representation of the overall ensemble and serving as a starting point for the search-based exploration presented in this paper.

We demonstrated above that our model performs well on the search task, clearly outperforming other problem-agnostic approaches and even performing well compared to domains-specific methods (see Sec. 7). Nevertheless, there is room for improvement. While the model produces consistent results across different training runs, it has quite a high variance wrt. random queries (Sec. 8.4). There are several reasons for this. First, the problem itself is uncertain: the goal is to detect high-level behavior "types", which can have rather vague boundaries. For example, for some members of the cylinder dataset, it is difficult to say at what exact point the transition to turbulence occurs. Another reason is related: we are not certain that the random queries express the behavior that we are trying to find. Indeed, we see that some queries lead to better results than others, but we found that a common cause of poor results is an "unclear" query. The model's response might be reasonable, but it doesn't align with our expectations. Though, the user can improve the query in such cases by providing additional examples. Yet another aspect is that there is a disconnect between the evaluation and the target application. During the evaluation, we sample query patches independently, while in a practical scenario, the next patch included by the user is conditioned on the previous results. In other words, the user adjusts the query based on the intermediate results, e.g., to filter out a false positive. We do not model this effect in our evaluation metrics for simplicity, but it should be done in the future, iteratively constructing queries by including the incorrectly ranked patches into the query.

Overall, we see great promise in applying self-supervised learning to the visual analysis of scientific data. Scientific data is usually unlabeled, but it is often large and has enough structure and metadata (coordinates, simulation parameters, multiple fields, etc.) to set up a pretext task for training. In this work, we proposed an approach that can be a starting point for many other research directions, both generic and domain-specific. Other model architectures could be exploited to improve the performance or express additional invariances, e.g., matching networks [50]. Also, many different self-supervised tasks can be explored: autoencoders, temporal prediction, variable reconstruction, and their combinations.

Another interesting direction for an extension is interpretable ML. Providing additional insight into the latent vectors could strengthen our method. One approach would be to add additional constraints to the model. For instance, we can use spatial attention to attribute different patch regions to latent dimensions and distances between the vectors. Similarly, sparsity [51] and latent distribution constraints [52] could be utilized to disentagle the latent dimensions. Another approach would be to do feature visualization, computing relevance scores [53] or visualizing input patches that activate specific latent dimensions the most [54]. Both ideas could even be combined, visualizing the model's features while constraining them to be more interpretable.

10 Conclusion

In this paper, we presented an approach to support the visual analysis of scientific data by constructing a spatiotemporal similarity metric using self-supervised machine learning. We showed that this metric allows for interactive search of similar behavior in ensemble data. The metric was able to produce high accuracy results on both experimental and simulation data, comparing well to manual domain-specific results and even discovering new findings on well-studied data. Overall, we see a lot of potential for self-supervised machine learning in scientific visualization, with many exciting directions for further research.

Acknowledgments

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2075 (SimTech) – 390740016.

References

[1] J. Wang, S. Hazarika, C. Li, and H. Shen, "Visualization and Visual Analysis of Ensemble Data: A Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 9, pp. 2853–2872, Sep. 2019.

[2] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning Fine-Grained Image Similarity with Deep Ranking," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1386–1393.

[3] H. Obermaier and K. I. Joy, "Future challenges for ensemble visualization," *IEEE Computer Graphics and Applications*, vol. 34, no. 3, pp. 8–11, May 2014.

[4] K. Potter, A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, and C. R. Johnson, "Ensemble-Vis: A Framework for the Statistical Visualization of Ensemble Data," in *In Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, 2009, pp. 233–240.

[5] J. Sanyal, S. Zhang, J. Dyer, A. Mercer, P. Amburn, and R. Moorhead, "Noodles: A Tool for Visualization of Numerical Weather Model Ensemble Uncertainty," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1421–1430, Nov. 2010.

[6] J. Waser, R. Fuchs, H. Ribicic, B. Schindler, G. Bloschl, and E. Groller, "World Lines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1458–1467, Nov. 2010.

[7] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller, "Visual Parameter Space Analysis: A Conceptual Framework," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2161–2170, Dec. 2014.

[8] S. Bruckner and T. Moeller, "Result-Driven Exploration of Simulation Parameter Spaces for Visual Effects Design," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1468–1476, Nov. 2010.

[9] M. Hummel, H. Obermaier, C. Garth, and K. I. Joy, "Comparative Visual Analysis of Lagrangian Transport in CFD Ensembles," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2743–2752, Dec. 2013.

[10] T.-H. Wei, C.-M. Chen, J. Woodring, H. Zhang, and H.-W. Shen, "Efficient distribution-based feature search in multi-field datasets," in *2017 IEEE Pacific Visualization Symposium (PacificVis)*, Apr. 2017, pp. 121–130.

[11] A. Kumpf, M. Rautenhaus, M. Riemer, and R. Westermann, "Visual Analysis of the Temporal Evolution of Ensemble Forecast Sensitivities," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 98–108, Jan. 2019.

[12] M. Jarema, I. Demir, J. Kehrer, and R. Westermann, "Comparative visual analysis of vector field ensembles," in *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct. 2015, pp. 81–88.

[13] Z. Wang, H. P. Seidel, and T. Weinkauf, "Multi-field Pattern Matching based on Sparse Feature Sampling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 807–816, Jan. 2016.

[14] L. Hao, C. G. Healey, and S. A. Bass, "Effective Visualization of Temporal Ensembles," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 787–796, Jan. 2016.

[15] W. He, H. Guo, H.-W. Shen, and T. Peterka, "eFESTA: Ensemble Feature Exploration with Surface Density Estimates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 4, pp. 1716–1731, Apr. 2020.

[16] A. Fofonov and L. Linsen, "Projected Field Similarity for Comparative Visualization of Multi-Run Multi-Field Time-Varying Spatial Data," *Computer Graphics Forum*, vol. 38, no. 1, pp. 286–299, Feb. 2019.

[17] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," *arXiv:1905.05055 [cs]*, May 2019.

[18] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A Survey of Deep Learning-Based Object Detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019.

[19] W. Li, R. Zhao, T. Xiao, and X. Wang, "DeepReID: Deep Filter Pairing Neural Network for Person Re-identification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 152–159.

[20] E. Ahmed, M. Jones, and T. K. Marks, "An improved deep learning architecture for person re-identification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 3908–3916.

[21] Q. Huang, W. Liu, and D. Lin, "Person Search in Videos with One Portrait Through Visual and Temporal Links," in *2018 European Conference on Computer Vision*, Cham, 2018, pp. 437–454.

[22] W. Han, P. Khorrami, T. L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T. S. Huang, "Seq-NMS for Video Object Detection," *arXiv:1602.08465 [cs]*, Aug. 2016.

[23] G. Bertasius, L. Torresani, and J. Shi, "Object Detection in Video with Spatiotemporal Sampling Networks," in *2018 European Conference on Computer Vision*, Cham, 2018, pp. 342–357.

[24] J. Deng, Y. Pan, T. Yao, W. Zhou, H. Li, and T. Mei, "Relation Distillation Networks for Video Object Detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 7022–7031.

[25] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative Unsupervised Feature Learning with Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 766–774.

[26] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification," in *2016 European Conference on Computer Vision*, Cham, 2016, pp. 527–544.

[27] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1422–1430.

[28] C. Doersch and A. Zisserman, "Multi-task Self-Supervised Visual Learning," *arXiv:1708.07860 [cs]*, Aug. 2017.

[29] K.-L. Ma, "Machine Learning to Boost the Next Generation of Visualization Technology," *IEEE Computer Graphics and Applications*, vol. 27, no. 5, pp. 6–9, Sep. 2007.

[30] A. Endert, W. Ribarsky, C. Turkay, B. W. Wong, I. Nabney, I. D. Blanco, and F. Rossi, "The state of the art in integrating machine learning into visual analytics," *Computer Graphics Forum*, 2017.

[31] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin, "Volume Upscaling with Convolutional Neural Networks," in *Proceedings of the Computer Graphics International Conference*, ser. CGI '17. New York, NY, USA: ACM, 2017, pp. 38:1–38:6.

[32] J. Han and C. Wang, "TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 205–215, 2020.

[33] M. Berger, J. Li, and J. A. Levine, "A Generative Model for Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 4, pp. 1636–1650, Apr. 2019.

[34] F. Hong, C. Liu, and X. Yuan, "DNN-VolVis: Interactive Volume Visualization Supported by Deep Neural Network," in *2019 IEEE Pacific Visualization Symposium (PacificVis)*, Apr. 2019, pp. 282–291.

[35] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. G. Nashed, and T. Peterka, "InSituNet: Deep Image Synthesis for Parameter Space Exploration of Ensemble Simulations," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019.

[36] W. He, J. Wang, H. Guo, H.-W. Shen, and T. Peterka, "CECAV-DNN: Collective ensemble comparison and visualization using deep neural networks," *Vis. Informatics*, vol. 4, no. 2, pp. 109–121, 2020.

[37] J. Han, J. Tao, and C. Wang, "FlowNet: A Deep Learning Framework for Clustering and Selection of Streamlines and Stream Surfaces," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018.

[38] R. Guo, T. Fujiwara, Y. Li, K. M. Lima, S. Sen, N. K. Tran, and K.-L. Ma, "Comparative visual analytics for assessing medical records with sequence embedding," *Visual Informatics*, Apr. 2020.

[39] G. Tkachev, S. Frey, and T. Ertl, "Local Prediction Models for Spatiotemporal Volume Visualization," *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[40] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification Using a "Siamese" Time Delay Neural Network," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, ser. NIPS'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744.

[41] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, Jun. 2005, pp. 539–546 vol. 1.

[42] G. R. Koch, "Siamese Neural Networks for One-Shot Image Recognition," in *2015 ICML Deep Learning Workshop*, 2015.

[43] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014.

[44] A. Geppert, A. Terzis, G. Lamanna, M. Marengo, and B. Weigand, "A benchmark study for the crown-type splashing dynamics of one- and two-component droplet wall–film interactions," *Experiments in Fluids*, vol. 58, no. 12, p. 172, Nov. 2017.

[45] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[46] R. Flamary and N. Courty, "POT python optimal transport library," 2017.

[47] M. Cuturi, "Sinkhorn Distances: Lightspeed Computation of Optimal Transport," in *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 2292–2300.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[49] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 586–595.

[50] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching Networks for One Shot Learning," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 3630–3638.

[51] A. Makhzani and B. J. Frey, "K-Sparse autoencoders," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[52] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations (ICLR)*, 2017.

[53] S. Lapuschkin, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation," *PLoS ONE*, vol. 10, Jul. 2015.

[54] C. Olah, A. Mordvintsev, and L. Schubert, "Feature Visualization," *Distill*, vol. 2, no. 11, p. e7, Nov. 2017.

Gleb Tkachev received his Masters degree in computer science from the University of Stuttgart, Germany. He is a PhD student at the University of Stuttgart Visualization Research Center (VISUS). His current research interests are focused on combining visual computing and machine learning methods for the analysis of scientific data.

Steffen Frey received his PhD degree in computer science from the University of Stuttgart, Germany in 2014. He is an assistant professor at the Bernoulli Institute at the University of Groningen, Netherlands. His research interests are in visualization methods for increasingly large quantities of scientific data.

Thomas Ertl received the MS degree in computer science from the University of Colorado at Boulder and the PhD degree in theoretical astrophysics from the University of Tübingen. He is a full professor of computer science with the University of Stuttgart, Germany in the Visualization and Interactive Systems Institute (VIS) and the director of the Visualization Research Center (VISUS). His research interests include visualization, computer graphics, and human computer interaction.