
Learning Similarity Metrics for Numerical Simulations

Georg Kohl¹ Kiwon Um¹ Nils Thuerey¹

Abstract

We propose a neural network-based approach that computes a stable and generalizing metric (*LSiM*) to compare data from a variety of numerical simulation sources. We focus on scalar time-dependent 2D data that commonly arises from motion and transport-based partial differential equations (PDEs). Our method employs a Siamese network architecture that is motivated by the mathematical properties of a metric. We leverage a controllable data generation setup with PDE solvers to create increasingly different outputs from a reference simulation in a controlled environment. A central component of our learned metric is a specialized loss function that introduces knowledge about the correlation between single data samples into the training process. To demonstrate that the proposed approach outperforms existing metrics for vector spaces and other learned, image-based metrics, we evaluate the different methods on a large range of test data. Additionally, we analyze generalization benefits of an adjustable training data difficulty and demonstrate the robustness of *LSiM* via an evaluation on three real-world data sets.

1. Introduction

Evaluating computational tasks for complex data sets is a fundamental problem in all computational disciplines. Regular vector space metrics, such as the L^2 distance, were shown to be very unreliable (Wang et al., 2004; Zhang et al., 2018), and the advent of deep learning techniques with convolutional neural networks (CNNs) made it possible to more reliably evaluate complex data domains such as natural images, texts (Benajiba et al., 2018), or speech (Wang et al., 2018). Our central aim is to demonstrate the usefulness of

¹Department of Informatics, Technical University of Munich, Munich, Germany. Correspondence to: Georg Kohl <georg.kohl@tum.de>.

CNN-based evaluations in the context of numerical simulations. These simulations are the basis for a wide range of applications ranging from blood flow simulations to aircraft design. Specifically, we propose a novel learned simulation metric (*LSiM*) that allows for a reliable similarity evaluation of simulation data.

Potential applications of such a metric arise in all areas where numerical simulations are performed or similar data is gathered from observations. For example, accurate evaluations of existing and new simulation methods with respect to a known ground truth solution (Oberkampf et al., 2004) can be performed more reliably than with a regular vector norm. Another good example is weather data for which complex transport processes and chemical reactions make in-place comparisons with common metrics unreliable (Jolliffe & Stephenson, 2012). Likewise, the long-standing, open questions of turbulence (Moin & Mahesh, 1998; Lin et al., 1998) can benefit from improved methods for measuring the similarity and differences in data sets and observations.

In this work, we focus on field data, i.e., dense grids of scalar values, similar to images, which were generated with known partial differential equations (PDEs) in order to ensure the availability of ground truth solutions. While we focus on 2D data in the following to make comparisons with existing techniques from imaging applications possible, our approach naturally extends to higher dimensions. Every sample of this 2D data can be regarded a high dimensional vector, so metrics on the corresponding vector space are applicable to evaluate similarities. These metrics, in the following denoted as *shallow metrics*, are typically simple, element-wise functions such as L^1 or L^2 distances. Their inherent problem is that they cannot compare structures on different scales or contextual information.

Many practical problems require solutions over time and need a vast number of non-linear operations that often result in substantial changes of the solutions even for small changes of the inputs. Hence, despite being based on known, continuous formulations, these systems can be seen as *chaotic*. We illustrate this behavior in Fig. 1, where two smoke flows are compared to a reference simulation. A single simulation parameter was varied for these examples, and a visual inspection shows that smoke plume (a) is more similar to the reference. This matches the data generation

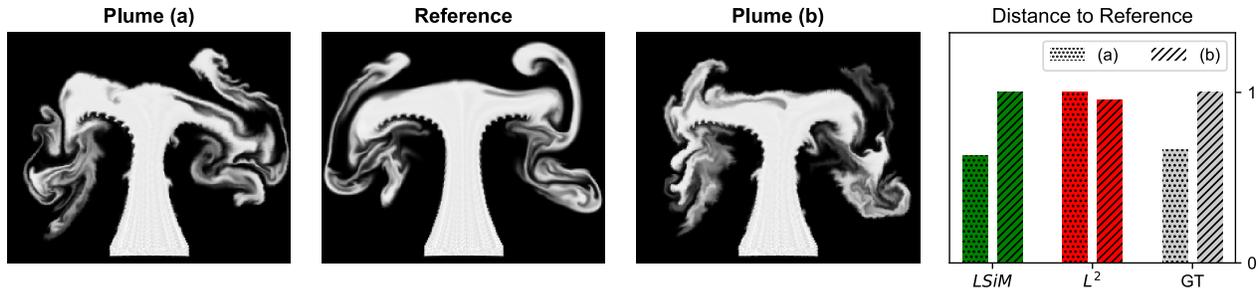


Figure 1. Example of field data from a fluid simulation of hot smoke with normalized distances for different metrics. Our method (*LSiM*, green) approximates the ground truth distances (GT, gray) determined by the data generation method best, i.e., version (a) is closer to the ground truth data than (b). An L^2 metric (red) erroneously yields a reversed ordering.

process: version (a) has a significantly smaller parameter change than (b) as shown in the inset graph on the right. *LSiM* robustly predicts the ground truth distances while the L^2 metric labels plume (b) as more similar. In our work, we focus on retrieving the relative distances of simulated data sets. Thus, we do not aim for retrieving the absolute parameter change but a relative distance that preserves ordering with respect to this parameter.

Using existing image metrics based on CNNs for this problem is not optimal either: natural images only cover a small fraction of the space of possible 2D data, and numerical simulation outputs are located in a fundamentally different data manifold within this space. Hence, there are crucial aspects that cannot be captured by purely learning from photographs. Furthermore, we have full control over the data generation process for simulation data. As a result, we can create arbitrary amounts of training data with gradual changes and a ground truth ordering. With this data, we can learn a metric that is not only able to directly extract and use features but also encodes interactions between them. The central contributions of our work are as follows:

- We propose a Siamese network architecture with feature map normalization, which is able to learn a metric that generalizes well to unseen motion and transport-based simulation methods.
- We propose a novel loss function that combines a correlation loss term with a mean squared error to improve the accuracy of the learned metric.
- In addition, we show how a data generation approach for numerical simulations can be employed to train networks with general and robust feature extractors for metric calculations.

Our source code, data sets, and final model are available at <https://github.com/tum-pbs/LSiM>.

2. Related Work

One of the earliest methods to go beyond using simple metrics based on L^p norms for natural images was the structural

similarity index (Wang et al., 2004). Despite improvements, this method can still be considered a shallow metric. Over the years, multiple large databases for human evaluations of natural images were presented, for instance, CSIQ (Larson & Chandler, 2010), TID2013 (Ponomarenko et al., 2015), and CID:IQ (Liu et al., 2014). With this data and the discovery that CNNs can create very powerful feature extractors that are able to recognize patterns and structures, deep feature maps quickly became established as means for evaluation (Amirshahi et al., 2016; Berardino et al., 2017; Bosse et al., 2016; Kang et al., 2014; Kim & Lee, 2017). Recently, these methods were improved by predicting the distribution of human evaluations instead of directly learning distance values (Prashnani et al., 2018; Talebi & Milanfar, 2018b). Zhang et al. compared different architecture and levels of supervision, and showed that metrics can be interpreted as a transfer learning approach by applying a linear weighting to the feature maps of any network architecture to form the image metric *LPIPS v0.1*. Typical use cases of these image-based CNN metrics are computer vision tasks such as detail enhancement (Talebi & Milanfar, 2018a), style transfer, and super-resolution (Johnson et al., 2016). Generative adversarial networks also leverage CNN-based losses by training a discriminator network in parallel to the generation task (Dosovitskiy & Brox, 2016).

Siamese network architectures are known to work well for a variety of comparison tasks such as audio (Zhang & Duan, 2017), satellite images (He et al., 2019), or the similarity of interior product designs (Bell & Bala, 2015). Furthermore, they yield robust object trackers (Bertinetto et al., 2016), algorithms for image patch matching (Hanif, 2019), and for descriptors for fluid flow synthesis (Chu & Thuerey, 2017). Inspired by these studies, we use a similar Siamese neural network architecture for our metric learning task. In contrast to other work on self-supervised learning that utilizes spatial or temporal changes to learn meaningful representations (Agrawal et al., 2015; Wang & Gupta, 2015), our method does not rely on tracked keypoints in the data.

While correlation terms have been used for learning joint representations by maximizing correlation of projected

views (Chandar et al., 2016) and are popular for style transfer applications via the Gram matrix (Ruder et al., 2016), they were not used for learning distance metrics. As we demonstrate below, they can yield significant improvements in terms of the inferred distances.

Similarity metrics for numerical simulations are a topic of ongoing investigation. A variety of specialized metrics have been proposed to overcome the limitations of L^p norms, such as the displacement and amplitude score from the area of weather forecasting (Keil & Craig, 2009) as well as permutation based metrics for energy consumption forecasting (Haben et al., 2014). Turbulent flows, on the other hand, are often evaluated in terms of aggregated frequency spectra (Pitsch, 2006). Crowd-sourced evaluations based on the human visual system were also proposed to evaluate simulation methods for physics-based animation (Um et al., 2017) and for comparing non-oscillatory discretization schemes (Um et al., 2019). These results indicate that visual evaluations in the context of field data are possible and robust, but they require extensive (and potentially expensive) user studies. Additionally, our method naturally extends to higher dimensions, while human evaluations inherently rely on projections with at most two spatial and one time dimension.

3. Constructing a CNN-based Metric

In the following, we explain our considerations when employing CNNs as evaluation metrics. For a comparison that corresponds to our intuitive understanding of distances, an underlying *metric* has to obey certain criteria. More precisely, a function $m : \mathbb{I} \times \mathbb{I} \rightarrow [0, \infty)$ is a metric on its input space \mathbb{I} if it satisfies the following properties $\forall x, y, z \in \mathbb{I}$:

$$m(x, y) \geq 0 \quad \text{non-negativity} \quad (1)$$

$$m(x, y) = m(y, x) \quad \text{symmetry} \quad (2)$$

$$m(x, y) \leq m(x, z) + m(z, y) \quad \text{triangle ineq.} \quad (3)$$

$$m(x, y) = 0 \iff x = y \quad \text{identity of indisc.} \quad (4)$$

The properties (1) and (2) are crucial as distances should be symmetric and have a clear lower bound. Eq. (3) ensures

that direct distances cannot be longer than a detour. Property (4), on the other hand, is not really useful for discrete operations as approximation errors and floating point operations can easily lead to a distance of zero for slightly different inputs. Hence, we focus on a relaxed, more meaningful definition $m(x, x) = 0 \forall x \in \mathbb{I}$, which leads to a so-called *pseudometric*. It allows for a distance of zero for different inputs but has to be able to spot identical inputs.

We realize these requirements for a pseudometric with an architecture that follows popular perceptual metrics such as *LPIPS*: The activations of a CNN are compared in latent space, accumulated with a set of weights, and the resulting per-feature distances are aggregated to produce a final distance value. Fig. 2 gives a visual overview of this process.

To show that the proposed Siamese architecture by construction qualifies as a pseudometric, the function

$$m(x, y) = m_2(m_1(x), m_1(y))$$

computed by our network is split into two parts: $m_1 : \mathbb{I} \rightarrow \mathbb{L}$ to compute the latent space embeddings $\tilde{x} = m_1(x)$, $\tilde{y} = m_1(y)$ from each input, and $m_2 : \mathbb{L} \rightarrow [0, \infty)$ to compare these points in the latent space \mathbb{L} . We chose operations for m_2 such that it forms a metric $\forall \tilde{x}, \tilde{y} \in \mathbb{L}$. Since m_1 always maps to \mathbb{L} , this means m has the properties (1), (2), and (3) on \mathbb{I} for any possible mapping m_1 , i.e., only a metric on \mathbb{L} is required. To achieve property (4), m_1 would need to be injective, but the compression of typical feature extractors precludes this. However, if m_1 is deterministic $m(x, x) = 0 \forall x \in \mathbb{I}$ is still fulfilled since identical inputs result in the same point in latent space and thus a distance of zero. More details for this proof can be found in App. A.

3.1. Base Network

The sole purpose of the base network (Fig. 2, in purple) is to extract feature maps from both inputs. The Siamese architecture implies that the weights of the base network are shared for both inputs, meaning all feature maps are comparable. We experimented with the feature extracting layers from var-

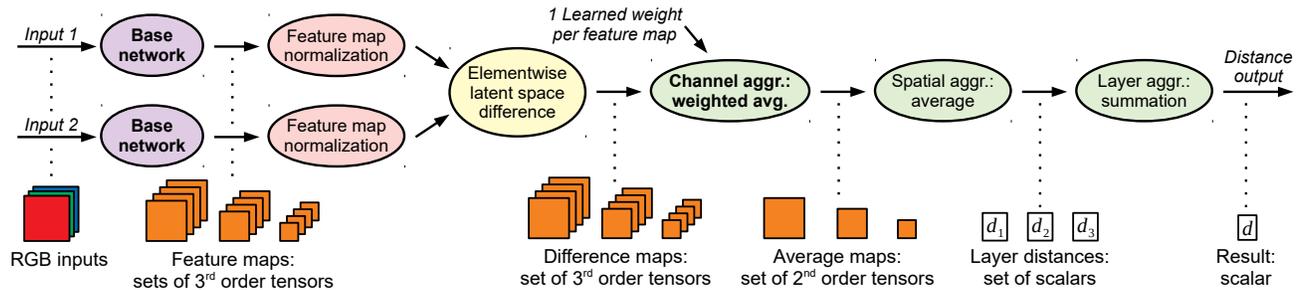


Figure 2. Overview of the proposed distance computation for a simplified base network that contains three layers with four feature maps each in this example. The output shape for every operation is illustrated below the transitions in orange and white. Bold operations are learned, i.e., contain weights influenced by the training process.

ious CNN architectures, such as AlexNet (Krizhevsky et al., 2017), VGG (Simonyan & Zisserman, 2015), SqueezeNet (Iandola et al., 2016), and a fluid flow prediction network (Thuerey et al., 2018). We considered three variants of these networks: using the original pre-trained weights, fine-tuning them, or re-training the full networks from scratch. In contrast to typical CNN tasks where only the result of the final output layer is further processed, we make use of the full range of extracted features across the layers of a CNN (see Fig. 2). This implies a slightly different goal compared to regular training: while early features should be general enough to allow for extracting more complex features in deeper layers, this is not their sole purpose. Rather, features in earlier layers of the network can directly participate in the final distance calculation and can yield important cues.

We achieved the best performance for our data sets using a base network architecture with five layers, similar to a reduced AlexNet, that was trained from scratch (see App. B.1). This feature extractor is fully convolutional and thus allows for varying spatial input dimensions, but for comparability to other models we keep the input size constant at 224×224 for our evaluation. In separate tests with interpolated inputs, we found that the metric still works well for scaling factors in the range $[0.5, 2]$.

3.2. Feature Map Normalization

The goal of normalizing the feature maps (Fig. 2, in red) is to transform the extracted features of each layer, which typically have very different orders of magnitude, into comparable ranges. While this task could potentially be performed by the learned weights, we found the normalization to yield improved performance in general.

Let \mathbf{G} denote a 4th order feature tensor with dimensions (g_b, g_c, g_x, g_y) from one layer of the base network. We form a series $\mathbf{G}_0, \mathbf{G}_1, \dots$ for every possible content of this tensor across our training samples. The normalization only happens in the channel dimension, so all following operations accumulate values along the dimension of g_c while keeping g_b, g_x , and g_y constant, i.e., are applied independently of the batch and spatial dimensions. The unit length normalization proposed by Zhang et al., i.e.,

$$\text{norm}_{\text{unit}}(\mathbf{G}) = \mathbf{G} / \|\mathbf{G}\|_2,$$

only considers the current sample. In this case, $\|\mathbf{G}\|_2$ is a 3rd order tensor with the Euclidean norms of \mathbf{G} along the channel dimension. Effectively, this results in a cosine distance, which only measures angles of the latent space vectors. To consider the vector magnitude, the most basic idea is to use the maximum norm of other training samples, and this leads to a global unit length normalization

$$\text{norm}_{\text{global}}(\mathbf{G}) = \mathbf{G} / \max(\|\mathbf{G}_0\|_2, \|\mathbf{G}_1\|_2, \dots).$$

Now, the magnitude of the current sample can be compared to other feature vectors, but this is not robust since the largest feature vector could be an outlier with respect to the typical content. Instead, we individually transform each component of a feature vector with dimension g_c to a standard normal distribution. This is realized by subtracting the mean and dividing by the standard deviation of all features element-wise along the channel dimension as follows:

$$\text{norm}_{\text{dist}}(\mathbf{G}) = \frac{1}{\sqrt{g_c - 1}} \frac{\mathbf{G} - \text{mean}(\mathbf{G}_0, \mathbf{G}_1, \dots)}{\text{std}(\mathbf{G}_0, \mathbf{G}_1, \dots)}.$$

These statistics are computed via a preprocessing step over the training data and stay fixed during training, as we did not observe significant improvements with more complicated schedules such as keeping a running mean. The magnitude of the resulting normalized vectors follows a chi distribution with $k = g_c$ degrees of freedom, but computing its mean $\sqrt{2} \Gamma((k+1)/2) / \Gamma(k/2)$ is expensive¹, especially for larger k . Instead, the mode of the chi distribution $\sqrt{g_c - 1}$ that closely approximates its mean is employed to achieve a consistent average magnitude of about one independently of g_c . As a result, we can measure angles for the latent space vectors and compare their magnitude in the global length distribution across all layers.

3.3. Latent Space Differences

Computing the difference of two latent space representations $\tilde{\mathbf{x}}, \tilde{\mathbf{y}} \in \mathbb{L}$ that consist of all extracted features from the two inputs $\mathbf{x}, \mathbf{y} \in \mathbb{I}$ lies at the core of the metric. This difference operator in combination with the following aggregations has to ensure that the metric properties above are upheld with respect to \mathbb{L} . Thus, the most obvious approach to employ an element-wise difference $\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i \forall i \in \{0, 1, \dots, \dim(\mathbb{L})\}$ is not suitable, as it invalidates non-negativity and symmetry. Instead, exponentiation of an absolute difference via $|\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|^p$ yields an L^p metric on \mathbb{L} , when combined with the correct aggregation and a p th root. $|\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|^2$ is used to compute the difference maps (Fig. 2, in yellow), as we did not observe significant differences for other values of p .

Considering the importance of comparing the extracted features, this simple feature difference does not seem optimal. Rather, one can imagine that improvements in terms of comparing one set of feature activations could lead to overall improvements for derived metrics. We investigated replacing these operations with a pre-trained CNN-based metric for each feature map. This creates a recursive process or “meta-metric” that reformulates the initial problem of learning input similarities in terms of learning feature space similarities. However, as detailed in App. B.3, we did not find any substantial improvements with this recursive approach. This implies that once a large enough number of expressive

¹ Γ denotes the gamma function for factorials

features is available for comparison, the in-place difference of each feature is sufficient to compare two inputs.

3.4. Aggregations

The subsequent aggregation operations (Fig. 2, in green) are applied to the difference maps to compress the contained per feature differences along the different dimensions into a single distance value. A simple summation in combination with an absolute difference $|\tilde{x}_i - \tilde{y}_i|$ above leads to an L^1 distance on the latent space \mathbb{L} . Similarly, we can show that average or learned weighted average operations are applicable too (see App. A). In addition, using a p -th power for the latent space difference requires a corresponding root operation after all aggregations, to ensure the metric properties with respect to \mathbb{L} .

To aggregate the difference maps along the channel dimension, we found the weighted average proposed by Zhang et al. to work very well. Thus, we use one learnable weight to control the importance of a feature. The weight is a multiplier for the corresponding difference map before summation along the channel dimension, and is clamped to be non-negative. A negative weight would mean that a larger difference in this feature produces a smaller overall distance, which is not helpful. For regularization, the learned aggregation weights utilize dropout during training, i.e., are randomly set to zero with a probability of 50%. This ensures that the network cannot rely on single features only, but has to consider multiple features for a more stable evaluation.

For spatial and layer aggregation, functions such as a summation or averaging are sufficient and generally interchangeable. We experimented with more intricate aggregation functions, e.g., by learning a spatial average or determining layer importance weights dynamically from the inputs. When the base network is fixed and the metric only has very few trainable weights, this did improve the overall performance. But, with a fully trained base network, the feature extraction seems to automatically adopt these aspects making a more complicated aggregation unnecessary.

4. Data Generation and Training

Similarity data sets for natural images typically rely on changing already existing images with distortions, noise, or other operations and assigning ground truth distances according to the strength of the operation. Since we can control the data creation process for numerical simulations directly, we can generate large amounts of simulation data with increasing dissimilarities by altering the parameters used for the simulations. As a result, the data contains more information about the nature of the problem, i.e., which changes of the data distribution should lead to increased distances, than by applying modifications as a post-process.

4.1. Data Generation

Given a set of model equations, e.g., a PDE from fluid dynamics, typical solution methods consist of a solver that, given a set of boundary conditions, computes discrete approximations of the necessary differential operators. The discretized operators and the boundary conditions typically contain problem dependent parameters, which we collectively denote with $p_0, p_1, \dots, p_i, \dots$ in the following. We only consider time dependent problems, and our solvers start with initial conditions at t_0 to compute a series of time steps t_1, t_2, \dots until a target point in time (t_t) is reached. At that point, we obtain a reference output field o_0 from one of the PDE variables, e.g., a velocity.

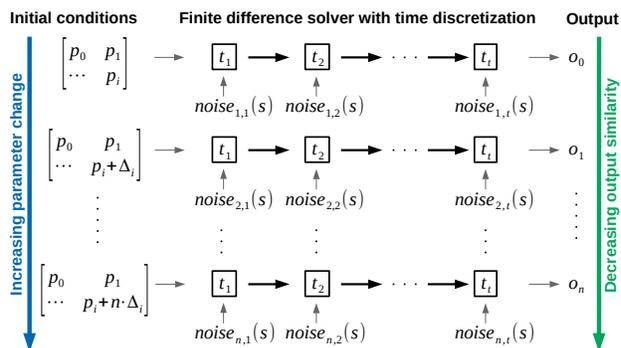


Figure 3. General data generation method from a PDE solver for a time dependent problem. With increasing changes of the initial conditions for a parameter p_i in Δ_i increments, the outputs decrease in similarity. Controlled Gaussian noise is injected in a simulation field of the solver. The difficulty of the learning task can be controlled by scaling Δ_i as well as the noise variance v .

For data generation, we incrementally change a single parameter p_i in n steps $\Delta_i, 2 \cdot \Delta_i, \dots, n \cdot \Delta_i$ to create a series of n outputs o_1, o_2, \dots, o_n . We consider a series obtained in this way to be increasingly different from o_0 . To create natural variations of the resulting data distributions, we add Gaussian noise fields with zero mean and adjustable variance v to an appropriate simulation field such as a velocity. This noise allows us to generate a large number of varied data samples for a single simulation parameter p_i . Furthermore, v serves as an additional parameter that can be varied in isolation to observe the same simulation with different levels of interference. This is similar in nature to numerical errors introduced by discretization schemes. These perturbations enlarge the space covered by the training data, and we found that training networks with suitable noise levels improves robustness as we will demonstrate below. The process for data generation is summarized in Fig. 3.

As PDEs can model extremely complex and chaotic behaviour, there is no guarantee that the outputs always exhibit increasing dissimilarity with the increasing parameter change. This behaviour is what makes the task of similar-

ity assessment so challenging. Even if the solutions are essentially chaotic, their behaviour is not arbitrary but rather governed by the rules of the underlying PDE. For our data set, we choose the following range of representative PDEs: We include a pure Advection-Diffusion model (AD), and Burger’s equation (BE) which introduces an additional viscosity term. Furthermore, we use the full Navier-Stokes equations (NSE), which introduce a conservation of mass constraint. When combined with a deterministic solver and a suitable parameter step size, all these PDEs exhibit chaotic behaviour at small scales, and the medium to large scale characteristics of the solutions shift smoothly with increasing changes of the parameters p_i .

The noise amplifies the chaotic behaviour to larger scales and provides a controlled amount of perturbations for the data generation. This lets the network learn about the nature of the chaotic behaviour of PDEs without overwhelming it with data where patterns are not observable anymore. The latter can easily happen when Δ_i or v grow too large and produce essentially random outputs. Instead, we specifically target solutions that are difficult to evaluate in terms of a shallow metric. We heuristically select the smallest v and a suitable Δ_i such that the ordering of several random output samples with respect to their L^2 difference drops below a correlation value of 0.8. For the chosen PDEs, v was small enough to avoid deterioration of the physical behaviour especially due to the diffusion terms, but different means of adjusting the difficulty may be necessary for other data.

4.2. Training

For training, the 2D scalar fields from the simulations were augmented with random flips, 90° rotations, and cropping to obtain an input size of 224×224 every time they are used. Identical augmentations were applied to each field of one given sequence to ensure comparability. Afterwards, each input sequence is collectively normalized to the range $[0, 255]$. To allow for comparisons with image metrics and provide the possibility to compare color data and full velocity fields during inference, the metric uses three input channels. During training, the scalar fields are duplicated to each channel after augmentation. Unless otherwise noted, networks were trained with a batch size of 1 for 40 epochs with the Adam optimizer using a learning rate of 10^{-5} . To evaluate the trained networks on validation and test inputs, only a bilinear resizing and the normalization step is applied.

5. Correlation Loss Function

The central goal of our networks is to identify relative differences of input pairs produced via numerical simulations. Thus, instead of employing a loss that forces the network to only infer given labels or distance values, we train our networks to infer the ordering of a given sequence of simula-

tion outputs o_0, o_1, \dots, o_n . We propose to use the Pearson correlation coefficient (see Pearson, 1920), which yields a value in $[-1, 1]$ that measures the linear relationship between two distributions. A value of 1 implies that a linear equation describes their relationship perfectly. We compute this coefficient for a full series of outputs such that the network can learn to extract features that arrange this data series in the correct ordering. Each training sample of our network consists of every possible pair from the sequence o_0, o_1, \dots, o_n and the corresponding ground truth distance distribution $\mathbf{c} \in [0, 1]^{0.5(n+1)n}$ representing the parameter change from the data generation. For a distance prediction $\mathbf{d} \in [0, \infty)^{0.5(n+1)n}$ of our network for one sample, we compute the loss with:

$$L(\mathbf{c}, \mathbf{d}) = \lambda_1(\mathbf{c} - \mathbf{d})^2 + \lambda_2\left(1 - \frac{(\mathbf{c} - \bar{\mathbf{c}}) \cdot (\mathbf{d} - \bar{\mathbf{d}})}{\|\mathbf{c} - \bar{\mathbf{c}}\|_2 \|\mathbf{d} - \bar{\mathbf{d}}\|_2}\right) \quad (5)$$

Here, the mean of a distance vector is denoted by $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$ for ground truth and prediction, respectively. The first part of the loss is a regular MSE term, which minimizes the difference between predicted and actual distances. The second part is the Pearson correlation coefficient, which is inverted such that the optimization results in a maximization of the correlation. As this formulation depends on the length of the input sequence, the two terms are scaled to adjust their relative influence with λ_1 and λ_2 . For the training, we chose $n = 10$ variations for each reference simulation. If n should vary during training, the influence of both terms needs to be adjusted accordingly. We found that scaling both terms to a similar order of magnitude worked best in our experiments.

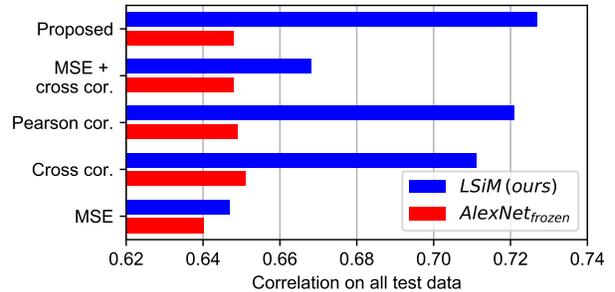


Figure 4. Performance comparison on our test data of the proposed approach (*LSiM*) and a smaller model (*AlexNet_{frozen}*) for different loss functions on the y-axis.

In Fig. 4, we investigate how the proposed loss function compares to other commonly used loss formulations for our full network and a pre-trained network, where only aggregation weights are learned. The performance is measured via Spearman’s rank correlation of predicted against ground truth distances on our combined test data sets. This is comparable to the A11 column in Tab. 1 and described in more

detail in Section 6.2. In addition to our full loss function, we consider a loss function that replaces the Pearson correlation with a simpler cross-correlation $(c \cdot d) / (\|c\|_2 \|d\|_2)$. We also include networks trained with only the MSE or only the correlation terms for each of the two variants.

A simple MSE loss yields the worst performance for both evaluated models. Using any correlation based loss function for the *AlexNet_{frozen}* metric (see Section 6.2) improves the results, but there is no major difference due to the limited number of only 1152 trainable weights. For *LSiM*, the proposed combination of MSE loss with the Pearson correlation performs better than using cross-correlation or only isolated Pearson correlation. Interestingly, combining cross correlation with MSE yields worse results than cross correlation by itself. This is caused by the cross correlation term influencing absolute distance values, which potentially conflicts with the MSE term. For our loss, the Pearson correlation only handles the relative ordering while the MSE deals with the absolute distances, leading to better inferred distances.

6. Results

In the following, we will discuss how the data generation approach was employed to create a large range of training and test data from different PDEs. Afterwards, the proposed metric is compared to other metrics, and its robustness is evaluated with several external data sets.

6.1. Data Sets

We created four training (*Smo*, *Liq*, *Adv* and *Bur*) and two test data sets (*LiqN* and *AdvD*) with ten parameter steps for each reference simulation. Based on two 2D NSE solvers, the smoke and liquid simulation training sets (*Smo* and *Liq*) add noise to the velocity field and feature varied initial conditions such as fluid position or obstacle properties, in addition to variations of buoyancy and gravity forces. The two other training sets (*Adv* and *Bur*) are based on 1D solvers for AD and BE, concatenated over time to form a 2D result. In both cases, noise was injected into the velocity field, and the varied parameters are changes to the field initialization and forcing functions.

For the test data set, we substantially change the data distribution by injecting noise into the density instead of the velocity field for AD simulations to obtain the *AdvD* data set and by including background noise for the velocity field of a liquid simulation (*LiqN*). In addition, we employed three more test sets (*Sha*, *Vid*, and *TID*) created without PDE models to explore the generalization for data far from our training data setup. We include a shape data set (*Sha*) that features multiple randomized moving rigid shapes, a video data set (*Vid*) consisting of frames from random video footage, and *TID2013* (Ponomarenko et al., 2015) as a perceptual image data set (*TID*). Below, we additionally list a combined correlation score (*All*) for all test sets apart from *TID*, which is excluded due to its different structure. Examples for each data set are shown in Fig. 5 and generation details with further samples can be found in App. D.

6.2. Performance Evaluation

To evaluate the performance of a metric on a data set, we first compute the distances from each reference simulation to all corresponding variations. Then, the predicted and the ground truth distance distributions over all samples are combined and compared using Spearman’s rank correlation coefficient (see Spearman, 1904). It is similar to the Pearson correlation, but instead it uses ranking variables, i.e., measures monotonic relationships of distributions.

The top part of Tab. 1 shows the performance of the shallow metrics L^2 and *SSIM* as well as the *LPIPS* metric (Zhang et al., 2018) for all our data sets. The results clearly show that shallow metrics are not suitable to compare the samples in our data set and only rarely achieve good correlation values. The perceptual *LPIPS* metric performs better in general and outperforms our method on the image data sets *Vid* and *TID*. This is not surprising as *LPIPS* is specifically trained for such images. For most of the simulation data sets, however, it performs significantly worse than for the image content. The last row of Tab. 1 shows the results of our *LSiM* model with a very good performance across all data sets and no negative outliers. Note that although it was not trained with any natural image content, it still performs well for the image test sets.

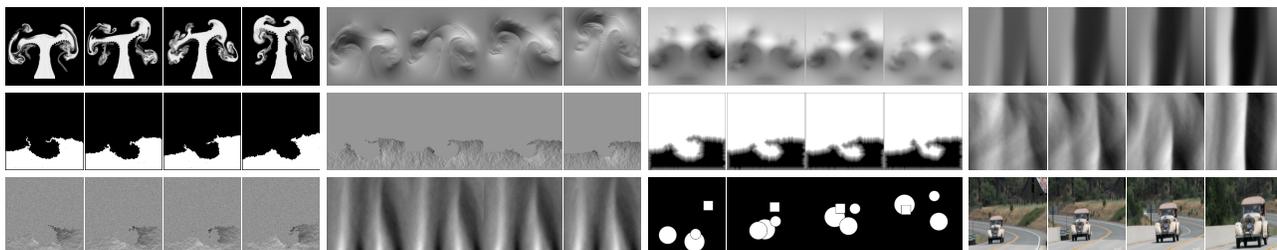
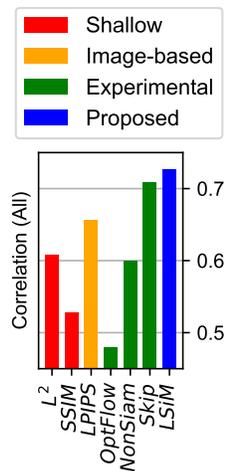


Figure 5. Samples from our data sets. For each subset the reference is on the left, and three variations in equal parameter steps follow. From left to right and top to bottom: *Smo* (density, velocity, and pressure), *Adv* (density), *Liq* (flags, velocity, and levelset), *Bur* (velocity), *LiqN* (velocity), *AdvD* (density), *Sha* and *Vid*.

Table 1. Performance comparison of existing metrics (top block), experimental designs (middle block), and variants of the proposed method (bottom block) on validation and test data sets measured in terms of Spearman’s rank correlation coefficient of ground truth against predicted distances. **Bold+underlined** values show the best performing metric for each data set, **bold** values are within a 0.01 error margin of the best performing, and *italic* values are 0.2 or more below the best performing. On the right, a visualization of the combined test data results is shown for selected models.

Metric	Validation data sets				Test data sets					
	Smo	Liq	Adv	Bur	TID	LiqN	AdvD	Sha	Vid	All
L^2	0.66	0.80	0.74	0.62	0.82	0.73	0.57	<i>0.58</i>	0.79	0.61
SSIM	0.69	0.73	0.77	0.71	0.77	<i>0.26</i>	0.69	<i>0.46</i>	0.75	<i>0.53</i>
LPIPS v0.1.	0.63	0.68	0.68	0.72	0.86	<i>0.50</i>	0.62	0.84	0.83	0.66
<i>AlexNet</i> _{random}	0.63	0.69	0.69	0.66	0.82	0.64	0.65	<i>0.67</i>	0.81	0.65
<i>AlexNet</i> _{frozen}	0.66	0.70	0.69	0.71	0.85	<i>0.40</i>	0.62	0.87	0.84	0.65
<i>Optical flow</i>	0.62	<i>0.57</i>	<i>0.36</i>	<i>0.37</i>	<i>0.55</i>	<i>0.49</i>	<i>0.28</i>	<i>0.61</i>	0.75	<i>0.48</i>
<i>Non-Siamese</i>	0.77	0.85	0.78	0.74	<i>0.65</i>	0.81	0.64	<i>0.25</i>	0.80	0.60
<i>Skip</i> _{from scratch}	0.79	0.83	0.80	0.74	0.85	0.78	0.61	0.78	0.83	0.71
<i>LSiM</i> _{noiseless}	0.77	0.77	0.76	0.72	0.85	0.62	0.58	0.86	0.82	0.68
<i>LSiM</i> _{strong noise}	0.65	<i>0.65</i>	0.67	0.69	0.84	<i>0.39</i>	0.54	0.89	0.82	0.64
<i>LSiM (ours)</i>	0.78	0.82	0.79	0.75	0.86	0.79	0.58	0.88	0.81	0.73



The middle block of Tab. 1 contains several interesting variants (more details can be found in App. B): *AlexNet*_{random} and *AlexNet*_{frozen} are small models, where the base network is the original AlexNet with pre-trained weights. *AlexNet*_{random} contains purely random aggregation weights without training, whereas *AlexNet*_{frozen} only has trainable weights for the channel aggregation and therefore lacks the flexibility to fully adjust to the data distribution of the numerical simulations. The random model performs surprisingly well in general, pointing to powers of the underlying Siamese CNN architecture.

Recognizing that many PDEs include transport phenomena, we investigated optical flow (Horn & Schunck, 1981) as a means to compute motion from field data. For the *Optical flow* metric, we used FlowNet2 (Ilg et al., 2016) to bidirectionally compute the optical flow field between two inputs and aggregate it to a single distance value by summing all flow vector magnitudes. On the data set Vid that is similar to the training data of FlowNet2, it performs relatively well, but in most other cases it performs poorly. This shows that computing a simple warping from one input to the other is not enough for a stable metric although it seems like an intuitive solution. A more robust metric needs the knowledge of the underlying features and their changes to generalize better to new data.

To evaluate whether a Siamese architecture is really beneficial, we used a *Non-Siamese* architecture that directly predicts the distance from both stacked inputs. For this purpose, we employed a modified version of AlexNet that reduces the weights of the feature extractor by 50% and of the remaining layers by 90%. As expected, this metric

works great on the validation data but has huge problems with generalization, especially on TID and Sha. In addition, even simple metric properties such as symmetry are no longer guaranteed because this architecture does not have the inherent constraints of the Siamese setup. Finally, we experimented with multiple fully trained base networks. As re-training existing feature extractors only provided small improvements, we used a custom base network with skip connections for the *Skip*_{from scratch} metric. Its results already come close to the proposed approach on most data sets.

The last block in Tab. 1 shows variants of the proposed approach trained with varied noise levels. This inherently changes the difficulty of the data. Hence, *LSiM*_{noiseless} was trained with relatively simple data without perturbations, whereas *LSiM*_{strong noise} was trained with strongly varying data. Both cases decrease the capabilities of the trained model on some of the validation and test sets. This indicates that the network needs to see a certain amount of variation at training time in order to become robust, but overly large changes hinder the learning of useful features (also see App. C).

6.3. Evaluation on Real-World Data

To evaluate the generalizing capabilities of our trained metric, we turn to three representative and publicly available data sets of captured and simulated real-world phenomena, namely buoyant flows, turbulence, and weather. For the former, we make use of the *ScalarFlow* data set (Eckert et al., 2019), which consists of captured velocities of buoyant scalar transport flows. Additionally, we include velocity data from the Johns Hopkins Turbulence Database (*JHTDB*)

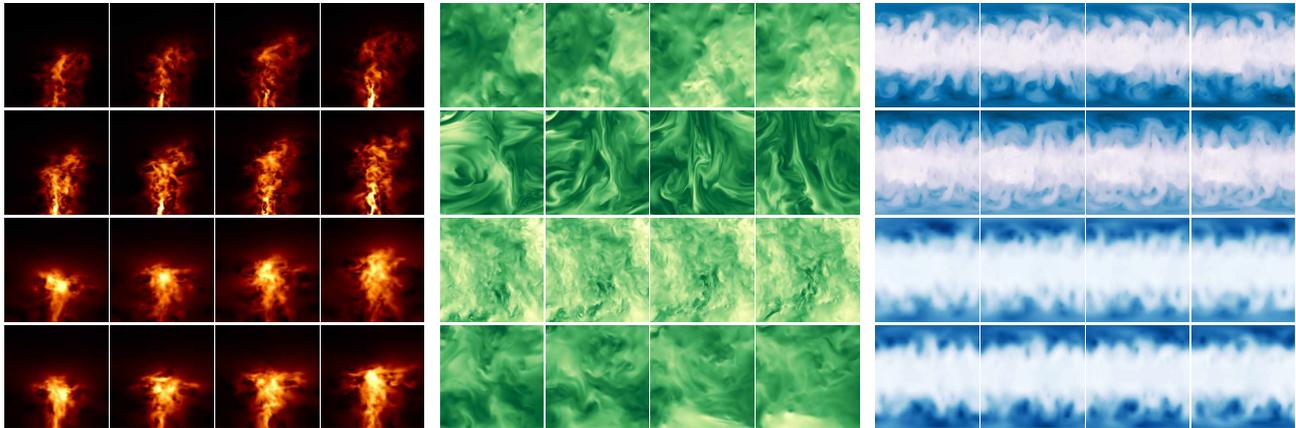


Figure 6. Examples from three real-world data repositories used for evaluation, visualized via color-mapping. Each block features four different sequences (rows) with frames in equal temporal or spatial intervals. Left: *ScalarFlow* – captured buoyant volumetric transport flows using the z-slice (top two) and z-mean (bottom two). Middle: *JHTDB* – four different turbulent DNS simulations. Right: *WeatherBench* – weather data consisting of temperature (top two) and geopotential (bottom two).

(Perlman et al., 2007), which represents direct numerical simulations of fully developed turbulence. As a third case, we use scalar temperature and geopotential fields from the *WeatherBench* repository (Rasp et al., 2020), which contains global climate data on a Cartesian latitude-longitude grid of the earth. Visualizations of this data via color-mapping the scalar fields or velocity magnitudes are shown in Fig. 6.

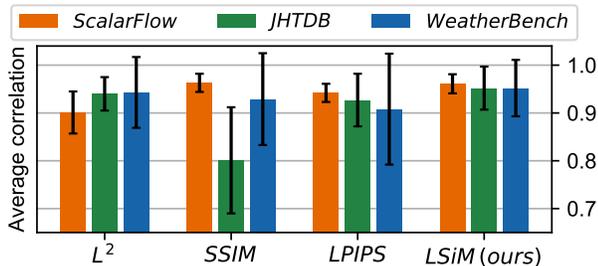


Figure 7. Spearman correlation values for multiple metrics on data from three repositories. Shown are mean and standard deviation over different temporal or spatial intervals used to create sequences.

For the results in Fig. 7, we extracted sequences of frames with fixed temporal and spatial intervals from each data set to obtain a ground truth ordering. Six different interval spacings for every data source are employed, and all velocity data is split by component. We then measure how well different metrics recover the original ordering in the presence of the complex changes of content, driven by the underlying physical processes. The *LSiM* model outlined in previous sections was used for inference without further changes.

Every metric is separately evaluated (see Section 6.2) for the six interval spacings with 180-240 sequences each. For *ScalarFlow* and *WeatherBench*, the data was additionally partitioned by z-slice or z-mean and temperature or geopo-

tential respectively, leading to twelve evaluations. Fig. 7 shows the mean and standard deviation of the resulting correlation values. Despite never being trained on any data from these data sets, *LSiM* recovers the ordering of all three cases with consistently high accuracy. It yields averaged correlations of 0.96 ± 0.02 , 0.95 ± 0.05 , and 0.95 ± 0.06 for *ScalarFlow*, *JHTDB*, and *WeatherBench*, respectively. The other metrics show lower means and higher uncertainty. Further details and results for the individual evaluations can be found in App. E.

7. Conclusion

We have presented the *LSiM* metric to reliably and robustly compare outputs from numerical simulations. Our method significantly outperforms existing shallow metric functions and provides better results than other learned metrics. We demonstrated the usefulness of the correlation loss, showed the benefits of a controlled data generation environment, and highlighted the stability of the obtained metric for a range of real-world data sets.

Our trained *LSiM* metric has the potential to impact a wide range of fields, including the fast and reliable accuracy assessment of new simulation methods, robust optimizations of parameters for reconstructions of observations, and guiding generative models of physical systems. Furthermore, it will be highly interesting to evaluate other loss functions, e.g., mutual information (Bachman et al., 2019) or contrastive predictive coding (Hénaff et al., 2019), and combinations with evaluations from perceptual studies (Um et al., 2019). We also plan to evaluate our approach for an even larger set of PDEs as well as for 3D and 4D data sets. Especially, turbulent flows are a highly relevant and interesting area for future work on learned evaluation metrics.

Acknowledgements

This work was supported by the ERC Starting Grant *realFlow* (StG-2015-637014). We would like to thank Stephan Rasp for preparing the *WeatherBench* data and all reviewers for helping to improve this work.

References

- Agrawal, P., Carreira, J., and Malik, J. Learning to see by moving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 37–45, 2015. doi:10.1109/ICCV.2015.13.
- Amirshahi, S. A., Pedersen, M., and Yu, S. X. Image Quality Assessment by Comparing CNN Features between Images. *Journal of Imaging Science and Technology*, 60(6), 2016. doi:10.2352/J.ImagingSci.Technol.2016.60.6.060410.
- Bachman, P., Hjelm, R. D., and Buchwalter, W. Learning representations by maximizing mutual information across views. *CoRR*, abs/1906.00910, 2019. URL <http://arxiv.org/abs/1906.00910>.
- Bell, S. and Bala, K. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics*, 34(4):98:1–98:10, 2015. doi:10.1145/2766959.
- Benajiba, Y., Sun, J., Zhang, Y., Jiang, L., Weng, Z., and Biran, O. Siamese networks for semantic pattern similarity. *CoRR*, abs/1812.06604, 2018. URL <http://arxiv.org/abs/1812.06604>.
- Berardino, A., Balle, J., Laparra, V., and Simoncelli, E. Eigen-Distortions of Hierarchical Representations. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, volume 30, 2017. URL <http://arxiv.org/abs/1710.02266>.
- Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. S. Fully-Convolutional Siamese Networks for Object Tracking. In *Computer Vision - ECCV 2016 Workshops, PT II*, volume 9914, pp. 850–865, 2016. doi:10.1007/978-3-319-48881-3_56.
- Bosse, S., Maniry, D., Mueller, K.-R., Wiegand, T., and Samek, W. Neural Network-Based Full-Reference Image Quality Assessment. In *2016 Picture Coding Symposium (PCS)*, 2016. doi:10.1109/PCS.2016.7906376.
- Chandar, S., Khapra, M. M., Larochelle, H., and Ravindran, B. Correlational neural networks. *Neural Computation*, 28(2): 257–285, 2016. doi:10.1162/NECO_a.00801.
- Chu, M. and Thuerey, N. Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors. *ACM Transactions on Graphics*, 36(4):69:1–69:14, 2017. doi:10.1145/3072959.3073643.
- Dosovitskiy, A. and Brox, T. Generating Images with Perceptual Similarity Metrics based on Deep Networks. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, volume 29, 2016. URL <http://arxiv.org/abs/1602.02644>.
- Eckert, M.-L., Um, K., and Thuerey, N. Scalarflow: A large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics*, 38(6), 2019. doi:10.1145/3355089.3356545.
- Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. URL <http://www.deeplearningbook.org>.
- Haben, S., Ward, J., Greetham, D. V., Singleton, C., and Grindrod, P. A new error measure for forecasts of household-level, high resolution electrical energy consumption. *International Journal of Forecasting*, 30(2):246–256, 2014. doi:10.1016/j.ijforecast.2013.08.002.
- Hanif, M. S. Patch match networks: Improved two-channel and Siamese networks for image patch matching. *Pattern Recognition Letters*, 120:54–61, 2019. doi:10.1016/j.patrec.2019.01.005.
- He, H., Chen, M., Chen, T., Li, D., and Cheng, P. Learning to match multitemporal optical satellite images using multi-support-patches Siamese networks. *Remote Sensing Letters*, 10(6):516–525, 2019. doi:10.1080/2150704X.2019.1577572.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi:10.1109/CVPR.2016.90.
- Hénaff, O. J., Razavi, A., Doersch, C., Eslami, S. M. A., and van den Oord, A. Data-efficient image recognition with contrastive predictive coding. *CoRR*, abs/1905.09272, 2019. URL <http://arxiv.org/abs/1905.09272>.
- Horn, B. K. and Schunck, B. G. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. doi:10.1016/0004-3702(81)90024-2.
- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017. doi:10.1109/CVPR.2017.243.
- Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. URL <http://arxiv.org/abs/1602.07360>.
- Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. FlowNet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016. URL <http://arxiv.org/abs/1612.01925>.
- Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Computer Vision - ECCV 2016, PT II*, volume 9906, pp. 694–711, 2016. doi:10.1007/978-3-319-46475-6_43.
- Jolliffe, I. T. and Stephenson, D. B. *Forecast verification: a practitioner's guide in atmospheric science*. John Wiley & Sons, 2012. doi:10.1002/9781119960003.
- Kang, L., Ye, P., Li, Y., and Doermann, D. Convolutional Neural Networks for No-Reference Image Quality Assessment. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1733–1740, 2014. doi:10.1109/CVPR.2014.224.
- Keil, C. and Craig, G. C. A displacement and amplitude score employing an optical flow technique. *Weather and Forecasting*, 24(5):1297–1308, 2009. doi:10.1175/2009WAF2222247.1.

- Kim, J. and Lee, S. Deep Learning of Human Visual Sensitivity in Image Quality Assessment Framework. In *30TH IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pp. 1969–1977, 2017. doi:10.1109/CVPR.2017.213.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. doi:10.1145/3065386.
- Larson, E. C. and Chandler, D. M. Most apparent distortion: full-reference image quality assessment and the role of strategy. *Journal of Electronic Imaging*, 19(1), 2010. doi:10.1117/1.3267105.
- Lin, Z., Hahm, T. S., Lee, W., Tang, W. M., and White, R. B. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, 281(5384):1835–1837, 1998. doi:10.1126/science.281.5384.1835.
- Liu, X., Pedersen, M., and Hardeberg, J. Y. CID:IQ - A New Image Quality Database. In *Image and Signal Processing, ICISP 2014*, volume 8509, pp. 193–202, 2014. doi:10.1007/978-3-319-07998-1_22.
- Moin, P. and Mahesh, K. Direct numerical simulation: a tool in turbulence research. *Annual review of fluid mechanics*, 30(1): 539–578, 1998. doi:10.1146/annurev.fluid.30.1.539.
- Oberkampf, W. L., Trucano, T. G., and Hirsch, C. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57:345–384, 2004. doi:10.1115/1.1767847.
- Pearson, K. Notes on the History of Correlation. *Biometrika*, 13 (1):25–45, 1920. doi:10.1093/biomet/13.1.25.
- Perlman, E., Burns, R., Li, Y., and Meneveau, C. Data exploration of turbulence simulations using a database cluster. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pp. 1–11, 2007. doi:10.1145/1362622.1362654.
- Pitsch, H. Large-eddy simulation of turbulent combustion. *Annu. Rev. Fluid Mech.*, 38:453–482, 2006. doi:10.1146/annurev.fluid.38.050304.092133.
- Ponomarenko, N., Jin, L., Ieremeiev, O., Lukin, V., Egiazarian, K., Astola, J., Vozel, B., Chehdi, K., Carli, M., Battisti, F., and Kuo, C. C. J. Image database TID2013: Peculiarities, results and perspectives. *Signal Processing-Image Communication*, 30: 57–77, 2015. doi:10.1016/j.image.2014.10.009.
- Prashnani, E., Cai, H., Mostofi, Y., and Sen, P. Picapp: Perceptual image-error assessment through pairwise preference. *CoRR*, abs/1806.02067, 2018. URL <http://arxiv.org/abs/1806.02067>.
- Rasp, S., Dueben, P., Scher, S., Weyn, J., Mouatadid, S., and Thuerey, N. Weatherbench: A benchmark dataset for data-driven weather forecasting. *CoRR*, abs/2002.00469, 2020. URL <http://arxiv.org/abs/2002.00469>.
- Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL <http://arxiv.org/abs/1505.04597>.
- Ruder, M., Dosovitskiy, A., and Brox, T. Artistic style transfer for videos. In *Pattern Recognition*, pp. 26–36, 2016. doi:10.1007/978-3-319-45886-1_3.
- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. URL <http://arxiv.org/abs/1409.1556>.
- Spearman, C. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. doi:10.2307/1412159.
- Talebi, H. and Milanfar, P. Learned Perceptual Image Enhancement. In *2018 IEEE International Conference on Computational Photography (ICCP)*, 2018a. doi:10.1109/ICCPHOT.2018.8368474.
- Talebi, H. and Milanfar, P. NIMA: Neural Image Assessment. *IEEE Transactions on Image Processing*, 27(8):3998–4011, 2018b. doi:10.1109/TIP.2018.2831899.
- Thuerey, N., Weissenow, K., Mehrotra, H., Mainali, N., Prantl, L., and Hu, X. Well, how accurate is it? A study of deep learning methods for reynolds-averaged navier-stokes simulations. *CoRR*, abs/1810.08217, 2018. URL <http://arxiv.org/abs/1810.08217>.
- Um, K., Hu, X., and Thuerey, N. Perceptual Evaluation of Liquid Simulation Methods. *ACM Transactions on Graphics*, 36(4), 2017. doi:10.1145/3072959.3073633.
- Um, K., Hu, X., Wang, B., and Thuerey, N. Spot the Difference: Accuracy of Numerical Simulations via the Human Visual System. *CoRR*, abs/1907.04179, 2019. URL <http://arxiv.org/abs/1907.04179>.
- Wang, X. and Gupta, A. Unsupervised learning of visual representations using videos. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2794–2802, 2015. doi:10.1109/ICCV.2015.320.
- Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi:10.1109/TIP.2003.819861.
- Wang, Z., Zhang, J., and Xie, Y. L2 Mispronunciation Verification Based on Acoustic Phone Embedding and Siamese Networks. In *2018 11TH International Symposium on Chinese Spoken Language Processing (ISCSLP)*, pp. 444–448, 2018. doi:10.1109/ISCSLP.2018.8706597.
- Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, 2018. doi:10.1109/CVPR.2018.00068.
- Zhang, Y. and Duan, Z. IMINET: Convolutional Semi-Siamese Networks for Sound Search by Vocal Imitation. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 304–308, 2017. doi:10.1109/TASLP.2018.2868428.
- Zhu, Y. and Bridson, R. Animating sand as a fluid. In *ACM SIGGRAPH 2005 Papers*, pp. 965–972, New York, NY, USA, 2005. doi:10.1145/1186822.1073298.

Appendix: Learning Similarity Metrics for Numerical Simulations

This supplemental document contains an analysis of the proposed metric design with respect to properties of metrics in general (App. A) and details to the used network architectures (App. B). Afterwards, material that deals with the data sets is provided. It contains examples and failure cases for each of the data domains and analyzes the impact of the data difficulty (App. C and D). Next, the evaluation on real-world data is described in more detail (App. E). Finally, we explore additional metric evaluations (App. F) and give an overview on the used notation (App. G).

The source code for using the trained *LSiM* metric and re-training the model from scratch are available at <https://github.com/tum-pbs/LSiM>. This includes the full data sets and the corresponding data generation scripts for the employed PDE solver.

A. Discussion of Metric Properties

To analyze if the proposed method qualifies as a *metric*, it is split in two functions $m_1 : \mathbb{I} \rightarrow \mathbb{L}$ and $m_2 : \mathbb{L} \times \mathbb{L} \rightarrow [0, \infty)$, which operate on the input space \mathbb{I} and the latent space \mathbb{L} . Through flattening elements from the input or latent space into vectors, $\mathbb{I} \simeq \mathbb{R}^a$ and $\mathbb{L} \simeq \mathbb{R}^b$ where a and b are the dimensions of the input data and all feature maps respectively, and both values have a similar order of magnitude. m_1 describes the non-linear function computed by the base network combined with the following normalization and returns a point in the latent space. m_2 uses two points in the latent space to compute a final distance value, thus it includes the latent space difference and the aggregation along the spatial, layer, and channel dimensions. With the Siamese network architecture, the resulting function for the entire approach is

$$m(\mathbf{x}, \mathbf{y}) = m_2(m_1(\mathbf{x}), m_1(\mathbf{y})).$$

The identity of indiscernibles mainly depends on m_1 because, even if m_2 itself guarantees this property, m_1 could still be non-injective, which means it can map different inputs to the same point in latent space $\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$ for $\mathbf{x} \neq \mathbf{y}$. Due to the complicated nature of m_1 , it is difficult to make accurate predictions about the injectivity of m_1 . Each base network layer of m_1 recursively processes the result of the preceding layer with various feature extracting operations. Here, the intuition is that significant changes in the input should produce different feature map results in one or more layers of the network. As very small changes in the input lead to zero valued distances predicted by the CNN (i.e., an

identical latent space for different inputs), m_1 is in practice not injective. In an additional experiment, the proposed architecture was evaluated on about 3500 random inputs from all our data sets, where the CNN received one unchanged and one slightly modified input. The modification consisted of multiple pixel adjustments by one bit (on 8-bit color images) in random positions and channels. When adjusting only a single pixel in the 224×224 input, the CNN predicts a zero valued distance on about 23% of the inputs, but we never observed an input where seven or more changed pixels resulted in a distance of zero in all experiments.

In this context, the problem of numerical errors is important because even two slightly different latent space representations could lead to a result that seems to be zero if the difference vanishes in the aggregation operations or is smaller than the floating point precision. On the other hand, an automated analysis to find points that have a different input but an identical latent space image is a challenging problem and left as future work.

The evaluation of the base network and the normalization is deterministic, and hence $\forall \mathbf{x} : m_1(\mathbf{x}) = m_1(\mathbf{x})$ holds. Furthermore, we know that $m(\mathbf{x}, \mathbf{x}) = 0$ if m_2 guarantees that $\forall m_1(\mathbf{x}) : m_2(m_1(\mathbf{x}), m_1(\mathbf{x})) = 0$. Thus, the remaining properties, i.e., non-negativity, symmetry, and the triangle inequality, only depend on m_2 since for them the original inputs are not relevant, but their respective images in the latent space. The resulting structure with a relaxed identity of indiscernibles is called a *pseudometric*, where $\forall \tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}} \in \mathbb{L}$:

$$m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \geq 0 \tag{6}$$

$$m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = m_2(\tilde{\mathbf{y}}, \tilde{\mathbf{x}}) \tag{7}$$

$$m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \leq m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{z}}) + m_2(\tilde{\mathbf{z}}, \tilde{\mathbf{y}}) \tag{8}$$

$$m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) = 0 \tag{9}$$

Notice that m_2 has to fulfill these properties with respect to the latent space but not the input space. If m_2 is carefully constructed, the metric properties still apply, independently of the actual design of the base network or the feature map normalization.

A first observation concerning m_2 is that if all aggregations were sum operations and the element-wise latent space difference was the absolute value of a difference operation, m_2 would be equivalent to computing the L^1 norm of the difference vector in latent space:

$$m_2^{sum}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \sum_{i=1}^b |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|.$$

Similarly, adding a square operation to the element-wise distance in the latent space and computing the square root at the very end leads to the L^2 norm of the latent space difference vector. In the same way, it is possible to use any L^p norm with the corresponding operations:

$$m_2^{sum}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\sum_{i=1}^b |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|^p \right)^{\frac{1}{p}}.$$

In both cases, this forms the metric induced by the corresponding norm, which by definition has all desired properties (6), (7), (8), and (9). If we change all aggregation methods to a weighted average operation, each term in the sum is multiplied by a weight w_i . This is even possible with learned weights, as they are constant at evaluation time if they are clamped to be positive as described above. Now, w_i can be attributed to both inputs by distributivity, meaning each input is element-wise multiplied with a constant vector before applying the metric, which leaves the metric properties untouched. The reason is that it is possible to define new vectors in the same space, equal to the scaled inputs. This renaming trivially provides the correct properties:

$$m_2^{weighted}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \sum_{i=1}^b w_i |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|,$$

$$\stackrel{w_i \geq 0}{=} \sum_{i=1}^b |w_i \tilde{\mathbf{x}}_i - w_i \tilde{\mathbf{y}}_i|.$$

Accordingly, doing the same with the L^p norm idea is possible, and each w_i just needs a suitable adjustment before distributivity can be applied, keeping the metric properties once again:

$$m_2^{weighted}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\sum_{i=1}^b w_i |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i|^p \right)^{\frac{1}{p}}$$

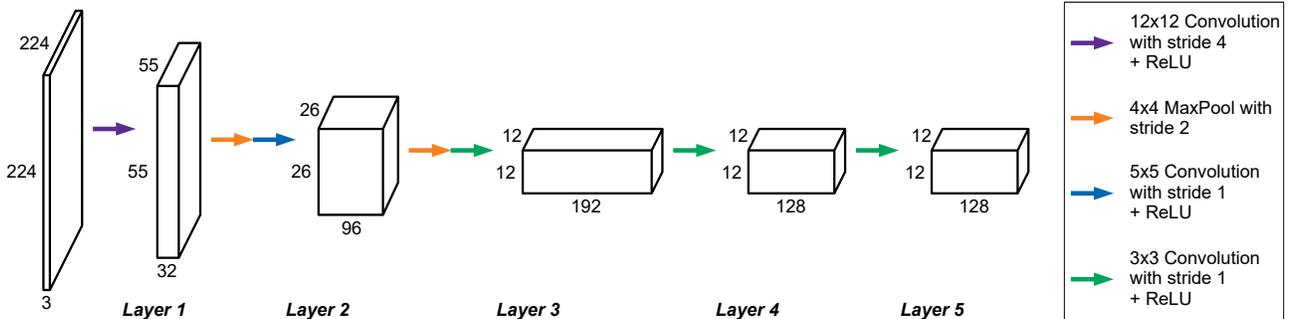


Figure 8. Proposed base network architecture consisting of five layers with up to 192 feature maps that are decreasing in spatial size. It is similar to the feature extractor from AlexNet as identical spatial dimensions for the feature maps are used, but it reduces the number of feature maps for each layer by 50% to have fewer weights.

$$= \left(\sum_{i=1}^b w_i |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| \dots |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| \right)^{\frac{1}{p}}$$

$$= \left(\sum_{i=1}^b w_i^{\frac{1}{p}} |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| w_i^{\frac{1}{p}} |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| \dots w_i^{\frac{1}{p}} |\tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i| \right)^{\frac{1}{p}},$$

$$\stackrel{w_i \geq 0}{=} \left(\sum_{i=1}^b |w_i^{\frac{1}{p}} \tilde{\mathbf{x}}_i - w_i^{\frac{1}{p}} \tilde{\mathbf{y}}_i|^p \right)^{\frac{1}{p}}.$$

With these weighted terms for m_2 , it is possible to describe all used aggregations and latent space difference methods. The proposed method deals with multiple higher order tensors instead of a single vector. Thus, the weights w_i additionally depend on constants such as the direction of the aggregations and their position in the latent space tensors. But it is easy to see that mapping a higher order tensor to a vector and keeping track of additional constants still retains all properties in the same way. As a result, the described architecture by design yields a pseudometric that is suitable for comparing simulation data in a way that corresponds to our intuitive understanding of distances.

B. Architectures

The following sections provide details regarding the architecture of the base network and some experimental design.

B.1. Base Network Design

Fig. 8 shows the architecture of the base network for the $LSiM$ metric. Its purpose is to extract features from both inputs of the Siamese architecture that are useful for the further processing steps. To maximise the usefulness and to avoid feature maps that show overly similar features, the chosen kernel size and stride of the convolutions are important. Starting with larger kernels and strides means the network has a big receptive field and can consider simple, low-level features in large regions of the input. For the two

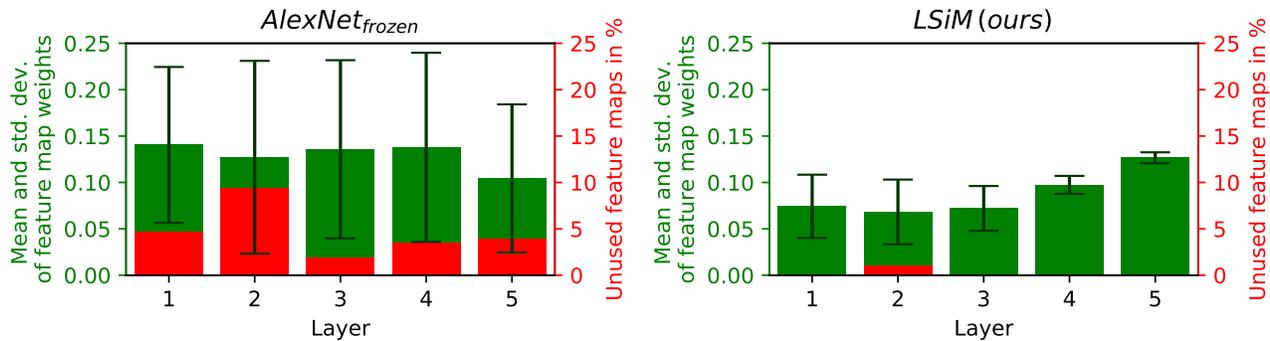


Figure 9. Analysis of the distributions of learned feature map aggregation weights across the base network layers. Displayed is a base network with pre-trained weights (left) in comparison to our method for fully training the base network (right). Note that the percentage of unused feature maps for most layers of our base network is 0%.

following layers, the large strides are replaced by additional MaxPool operations that serve a similar purpose and reduce the spatial size of the feature maps.

For the three final layers, only small convolution kernels and strides are used, but the number of channels is significantly larger than before. These deep features maps typically contain high-level structures, which are most important to distinguish complex changes in the inputs. Keeping the number of trainable weights as low as possible was an important consideration for this design to prevent overfitting to certain simulations types and increase generality. We explored a weight range by using the same architecture and only scaling the number of feature maps in each layer. The final design shown in Fig. 8 with about 0.62 million weights worked best for our experiments.

In the following, we analyze the contributions of the per-layer features of two different metric networks to highlight differences in terms of how the features are utilized for the distance estimation task. In Fig. 9, our *LSiM* network yields a significantly smaller standard deviation in the learned weights that aggregate feature maps of five layers, compared to a pre-trained base network. This means, all feature maps contribute to establishing the distances similarly, and the aggregation just fine-tunes the relative importance of each feature. In addition, almost all features receive a weight greater than zero, and as a result, more features are contributing to the final distance value.

Employing a fixed pre-trained feature extractor, on the other hand, shows a very different picture: Although the mean across the different network layers is similar, the contributions of different features vary strongly, which is visible in the standard deviation being significantly larger. Furthermore, 2-10% of the feature maps in each layer receive a weight of zero and hence were deemed not useful at all for establishing the distances. This illustrates the usefulness of a targeted network in which all features contribute to the distance inference.

B.2. Feature Map Normalization

In the following, we analyze how the different feature map normalizations discussed in Section 3.2 of the main paper affect the performance of our metric. We compare using no normalization $norm_{none}(\mathbf{G}) = \mathbf{G}$, the unit length normalization via division by the norm of a feature vector $norm_{unit}(\mathbf{G}) = \mathbf{G} / \|\mathbf{G}\|_2$ proposed by Zhang et al., a global unit length normalization $norm_{global}(\mathbf{G}) = \mathbf{G} / \max(\|\mathbf{G}_0\|_2, \|\mathbf{G}_1\|_2, \dots)$ that considers the norm of all feature vectors in the entire training set, and the proposed normalization to a scaled chi distribution

$$norm_{dist}(\mathbf{G}) = \frac{1}{\sqrt{g_c - 1}} \frac{\mathbf{G} - \text{mean}(\mathbf{G}_0, \mathbf{G}_1, \dots)}{\text{std}(\mathbf{G}_0, \mathbf{G}_1, \dots)}.$$

Fig. 10 shows a comparison of these normalization methods on the combined test data. Using no normalization is significantly detrimental to the performance of the metric as succeeding operations cannot reliably compare the features. A unit length normalization of a single sample is already a major improvement since following operations now have a predictable range of values to work with. This corresponds to a cosine distance, which only measures angles of the feature vectors and entirely neglects their length.

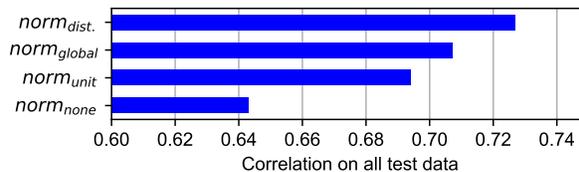


Figure 10. Performance on our test data for different feature map normalization approaches.

Using the maximum norm across all training samples (computed in a pre-processing step and fixed for training) introduces additional information as the network can now compare magnitudes as well. However, this comparison is not stable as the maximum norm can be an outlier with respect to the typical content of the corresponding feature.

The proposed normalization forms a chi distribution by individually transforming each component of the feature vector to a standard normal distribution. Afterwards, scaling with the inverse mode of the chi distribution leads to a consistent average magnitude close to one. It results in the best performing metric since both length and angle of the feature vectors can be reliably compared by the following operations.

B.3. Recursive “Meta-Metric”

Since comparing the feature maps is a central operation of the proposed metric calculations, we experimented with replacing it with an existing CNN-based metric. In theory, this would allow for a recursive, arbitrarily deep network that repeatedly invokes itself: first, the extracted representations of inputs are used and then the representations extracted from the previous representations, etc. In practice, however, using more than one recursion step is currently not feasible due to increasing computational requirements in addition to vanishing gradients.

Fig. 11 shows how our computation method can be modified for a CNN-based latent space difference, instead of an element-wise operation. Here we employ *LPIPS* (Zhang et al., 2018). There are two main differences compared to proposed method. First, the *LPIPS* latent space difference creates single distance values for a pair of feature maps instead of a spatial feature difference. As a result, the following aggregation is a single learned average operation and spatial or layer aggregations are no longer necessary. We also performed experiments with a spatial *LPIPS* version here, but due to memory limitations, these were not successful. Second, the convolution operations in *LPIPS* have a lower limit for spatial resolution, and some feature maps of our base network are quite small (see Fig. 8). Hence, we up-scale the feature maps below the required spatial size of 32×32 using nearest neighbor interpolation.

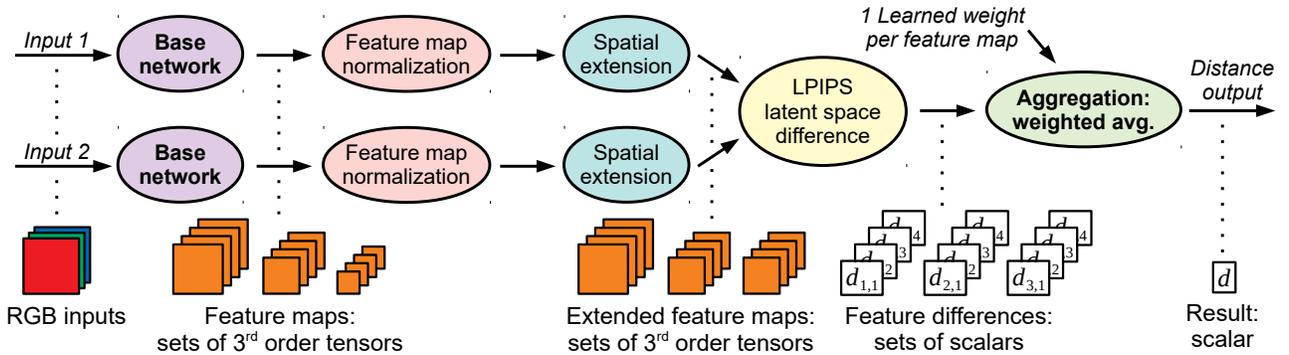


Figure 11. Adjusted distance computation for a *LPIPS*-based latent space difference. To provide sufficiently large inputs for *LPIPS*, small feature maps are spatially enlarged with nearest neighbor interpolation. In addition, *LPIPS* creates scalar instead of spatial differences leading to a simplified aggregation.

On our combined test data, such a metric with a fully trained base network achieves a performance comparable to *AlexNet_{random}* or *AlexNet_{frozen}*.

B.4. Optical Flow Metric

In the following, we describe our approach to compute a metric via optical flow (OF). For an efficient OF evaluation, we employed a pre-trained network (Ilg et al., 2016). From an OF network $f : \mathbb{I} \times \mathbb{I} \rightarrow \mathbb{R}^{i_{max} \times j_{max} \times 2}$ with two input data fields $\mathbf{x}, \mathbf{y} \in I$, we get the flow vector field $f^{\mathbf{x}\mathbf{y}}(i, j) = (f_1^{\mathbf{x}\mathbf{y}}(i, j), f_2^{\mathbf{x}\mathbf{y}}(i, j))^T$, where i and j denote the locations, and f_1 and f_2 denote the components of the flow vectors. In addition, we have a second flow field $f^{\mathbf{y}\mathbf{x}}(i, j)$ computed from the reversed input ordering. We can now define a function $m : \mathbb{I} \times \mathbb{I} \rightarrow [0, \infty)$:

$$m(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{i_{max}} \sum_{j=0}^{j_{max}} \sqrt{(f_1^{\mathbf{x}\mathbf{y}}(i, j))^2 + (f_2^{\mathbf{x}\mathbf{y}}(i, j))^2} + \sqrt{(f_1^{\mathbf{y}\mathbf{x}}(i, j))^2 + (f_2^{\mathbf{y}\mathbf{x}}(i, j))^2}.$$

Intuitively, this function computes the sum over the magnitudes of all flow vectors in both vector fields. With this definition, it is obvious that $m(\mathbf{x}, \mathbf{y})$ fulfills the metric properties of non-negativity and symmetry (see Eq. (6) and (7)). Under the assumption that identical inputs create a zero flow field, a relaxed identity of indiscernibles holds as well (see Eq. (9)). Compared to the proposed approach, there is no guarantee for the triangle inequality though, thus $m(\mathbf{x}, \mathbf{y})$ only qualifies as a pseudo-semimetric.

Fig. 12 shows flow visualizations on data examples produced by FlowNet2. The metric works relatively well for inputs that are similar to the training data from FlowNet2 such as the shape data example in the top row. For data that provides some outline, e.g., the smoke simulation example in the middle row or also liquid data, the metric does

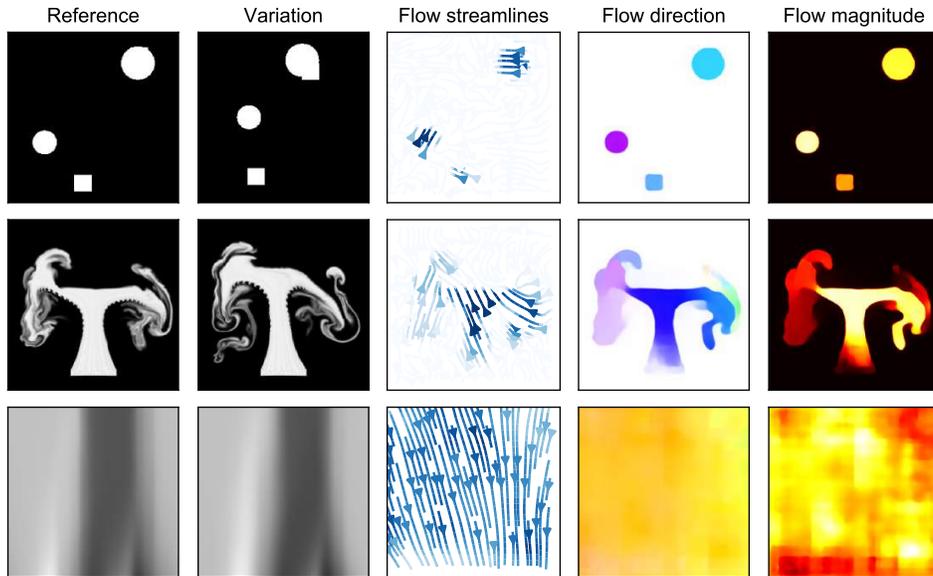


Figure 12. Outputs from FlowNet2 on data examples. The flow streamlines are sparse visualization of the resulting flow field and indicate the direction of the flow by their orientation and its magnitude by their color (darker being larger). The two visualizations on the right show the dense flow field and are color-coded to show the flow direction (blue/yellow: vertical, green/red: horizontal) and the flow magnitude (brighter being larger).

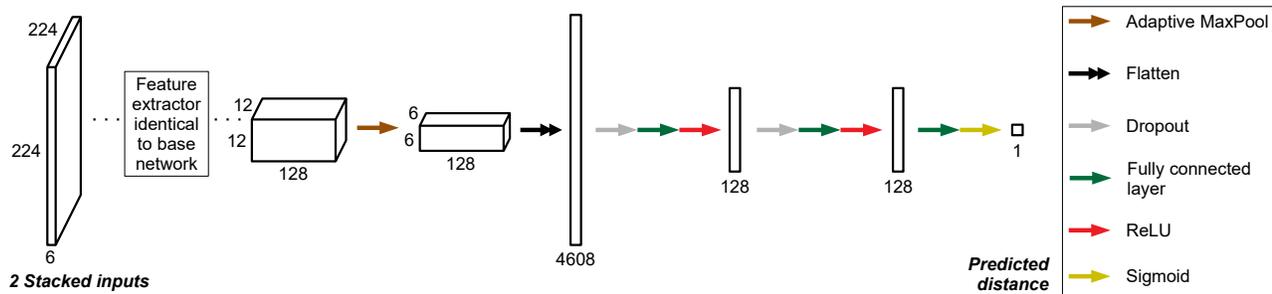


Figure 13. Non-Siamese network architecture with the same feature extractor used in Fig. 8. It uses both stacked inputs and directly predicts the final distance value from the last set of feature maps with several fully connected layers.

not work as well but still provides a reasonable flow field. However, for full spatial examples such as the Burger’s or Advection-Diffusion cases (see bottom row), the network is no longer able to produce meaningful flow fields. The results are often a very uniform flow with similar magnitude and direction.

B.5. Non-Siamese Architecture

To compute a metric without the Siamese architecture outlined above, we use a network structure with a single output as shown in Fig. 13. Thus, instead of having two identically feature extractors and combining the feature maps, here the distance is directly predicted from the stacked inputs with a single network with about 1.24 million weights. After using the same feature extractor as described in Section B.1, the final set of feature maps is spatially reduced with an adap-

tive MaxPool operation. Next, the result is flattened, and three consecutive fully connected layers process the data to form the final prediction. Here, the last activation function is a sigmoid instead of ReLU. The reason is that a ReLU would clamp every negative intermediate value to a zero distance, while a sigmoid compresses the intermediate value to a small distance that is more meaningful than directly clamping it.

In terms of metric properties, this architecture only provides non-negativity (see Eq. (6)) due to the final sigmoid function. All other properties cannot be guaranteed without further constraints. This is the main disadvantage of a non-Siamese network. These issues could be alleviated with specialized training data or by manually adding constraints to the model, e.g., to have some amount of symmetry (see Eq. (7)) and at least a weakened identity of indiscernibles (see Eq. (9)).

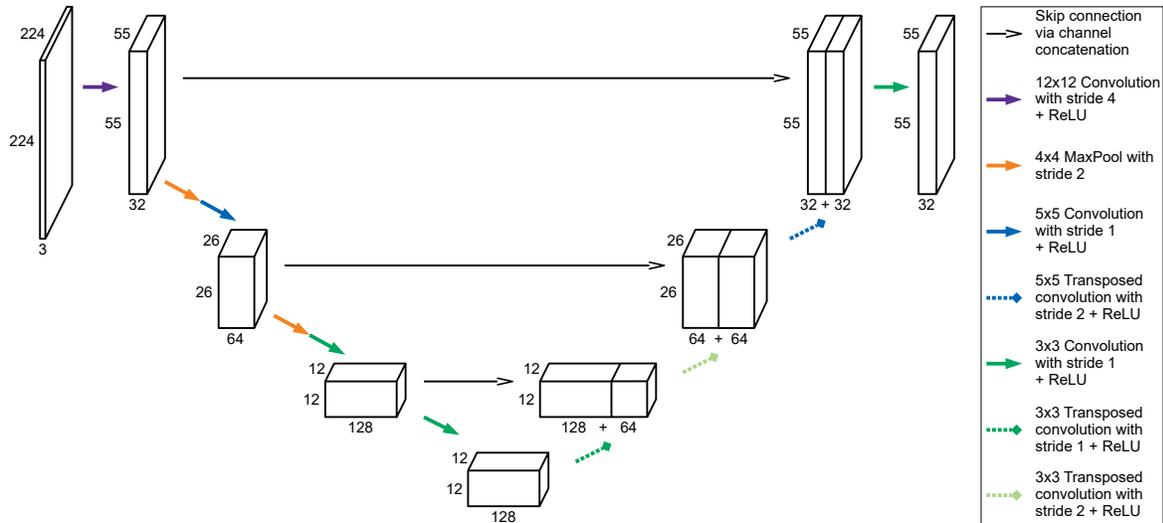


Figure 14. Network architecture with skip connections for better information transport between feature maps. Transposed convolutions are used to upscale the feature maps in the second half of the network to match the spatial size of earlier layers for the skip connections.

However, compared to a Siamese network that guarantees them by design, these extensions are clearly sub-optimal. As a result of the missing properties, this network has significant problems with generalization. While it performs well on the training data, the performance noticeably deteriorates for several of the test data sets.

B.6. Skip Connections in Base Network

As explained above, our base network primarily serves as a feature extractor to produce activations that are employed to evaluate a learned metric. In many state-of-the-art methods, networks with skip connections are employed (Ronneberger et al., 2015; He et al., 2016; Huang et al., 2017), as experiments have shown that these connections help to preserve information from the inputs. In our case, the classification “output” of a network such as the AlexNet plays no actual role. Rather, the features extracted along the way are crucial. Hence, skip connections should not improve the inference task for our metrics.

To verify that this is the case, we have included tests with a base network (see Fig. 14) similar to the popular UNet architecture (Ronneberger et al., 2015). For our experiments, we kept the early layers closely in line with the feature extractors that worked well for the base network (see Section B.1). Only the layers in the decoder part have an increased spatial feature map size to accommodate the skip connections. As expected, this network can be used to compute reliable metrics for the input data without negatively affecting the performance. However, as expected, the improvements of skip connections for regular inference tasks do not translate into improvements for the metric calculations.

C. Impact of Data Difficulty

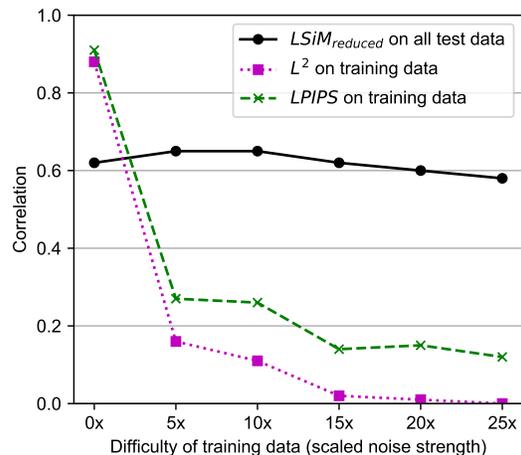


Figure 15. Impact of increasing data difficulty for a reduced training data set. Evaluations on training data for L^2 and $LPIPS$, and the test performance of models trained with the different reduced data sets ($LSiM_{reduced}$) are shown.

We shed more light on the aspect of noise levels and data difficulty via six reduced data sets that consist of a smaller amount of Smoke and Advection-Diffusion data with differently scaled noise strength values. Results are shown in Fig. 15. Increasing the noise level creates more difficult data as shown by the dotted and dashed plots representing the performance of the L^2 and the $LPIPS$ metric on each data set. Both roughly follow an exponentially decreasing function. Each point on the solid line plot is the test result of a reduced $LSiM$ model trained on the data set with the corresponding noise level. Apart from the data, the entire training

setup was identical. This shows that the training process is very robust to the noise, as the result on the test data only slowly decreases for very high noise levels. Furthermore, small amounts of noise improve the generalization compared to the model that was trained without any noise. This is somewhat expected, as a model that never saw noisy data during training cannot learn to extract features which are robust with respect to noise.

D. Data Set Details

In the following sections, the generation of each used data set is described. For each figure showing data samples (consisting of a reference simulation and several variants with a single changing initial parameter), the leftmost image is the reference and the images to the right show the variants in order of increasing parameter change. For the figures 16, 17, 18, and 19, the first subfigure (a) demonstrates that medium and large scale characteristics behave very non-chaotic for simulations without any added noise. They are only included for illustrative purposes and are not used for training. The second and third subfigure (b) and (c) in each case show the training data of *LSiM*, where the large majority of data falls into the category (b) of normal samples that follow the generation ordering, even with more varying behaviour. Category (c) is a small fraction of the training data, and the shown examples are specifically picked to show how the chaotic behaviour can sometimes override the ordering intended by the data generation in the worst case. Occasionally, category (d) is included to show how normal data samples from the test set differ from the training data.

D.1. Navier-Stokes Equations

These equations describe the general behaviour of fluids with respect to advection, viscosity, pressure, and mass conservation. Eq. (10) defines the conservation of momentum, and Eq. (11) constraints the conservation of mass:

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\frac{\nabla P}{\rho} + \nu \nabla^2 u + g, \quad (10)$$

$$\nabla \cdot u = 0. \quad (11)$$

In this context, u is the velocity, P is the pressure the fluid exerts, ρ is the density of the fluid (usually assumed to be constant), ν is the kinematic viscosity coefficient that indicates the thickness of the fluid, and g denotes the acceleration due to gravity. With this PDE, three data sets were created using a smoke and a liquid solver. For all data, 2D simulations were run until a certain step, and useful data fields were exported afterwards.

SMOKE

For the smoke data, a standard Eulerian fluid solver using a preconditioned pressure solver based on the conjugate

gradient method and Semi-Lagrangian advection scheme was employed.

The general setup for every smoke simulation consists of a rectangular smoke source at the bottom with a fixed additive noise pattern to provide smoke plumes with more details. Additionally, there is a downwards directed, spherical force field area above the source, which divides the smoke in two major streams along it. We chose this solution over an actual obstacle in the simulation in order to avoid overfitting to a clearly defined black obstacle area inside the smoke data. Once the simulation reaches a predefined time step, the density, pressure, and velocity fields (separated by dimension) are exported and stored. Some example sequences can be found in Fig. 16. With this setup, the following initial conditions were varied in isolation:

- Smoke buoyancy in x- and y-direction
- Strength of noise added to the velocity field
- Amount of force in x- and y-direction provided by the force field
- Orientation and size of the force field
- Position of the force field in x- and y-direction
- Position of the smoke source in x- and y-direction

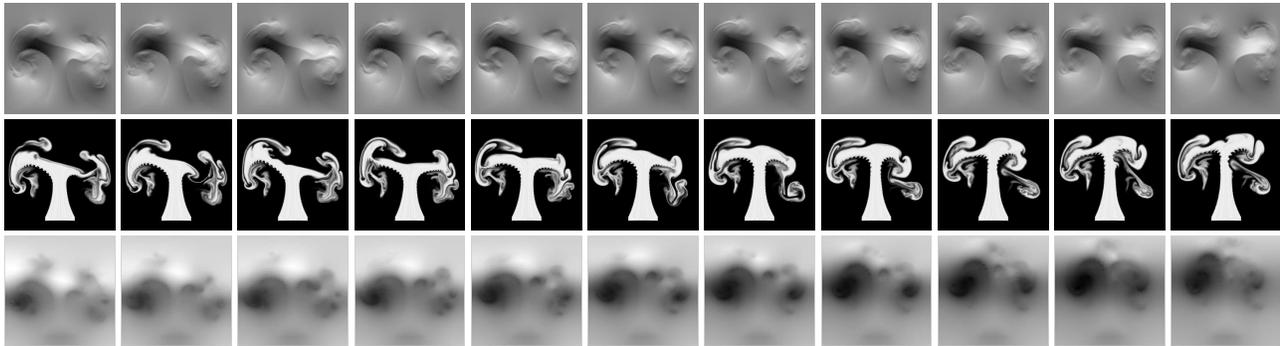
Overall, 768 individual smoke sequences were used for training, and the validation set contains 192 sequences with different initialization seeds.

LIQUID

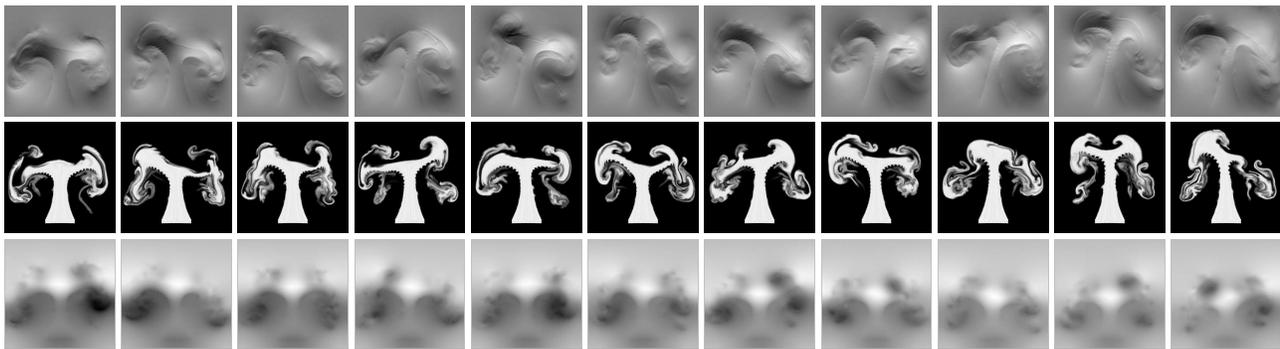
For the liquid data, a solver based on the fluid implicit particle (FLIP) method (Zhu & Bridson, 2005) was employed. It is a hybrid Eulerian-Lagrangian approach that replaces the Semi-Lagrangian advection scheme with particle based advection to reduce numerical dissipation. Still, this method is not optimal as we experienced problems such as mass loss, especially for larger noise values.

The simulation setup consists of a large breaking dam and several smaller liquid areas for more detailed splashes. After the dam hits the simulation boundary, a large, single drop of liquid is created in the middle of the domain that hits the already moving liquid surface. Then, the extrapolated level set values, binary indicator flags, and the velocity fields (separated by dimension) are saved. Some examples are shown in Fig. 17. The list of varied parameters include:

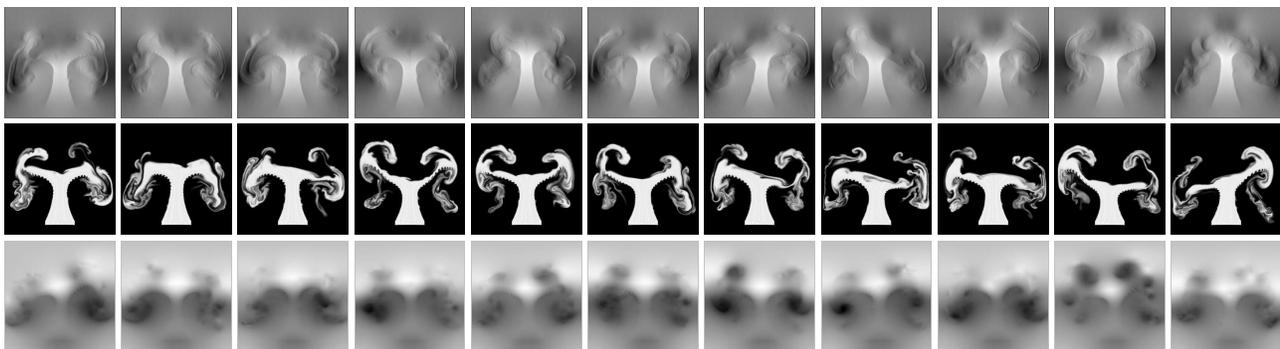
- Radius of the liquid drop
- Position of the drop in x- and y-direction
- Amount of additional gravity force in x- and y-direction
- Strength of noise added to the velocity field



(a) Data samples generated without noise: tiny output changes following generation ordering

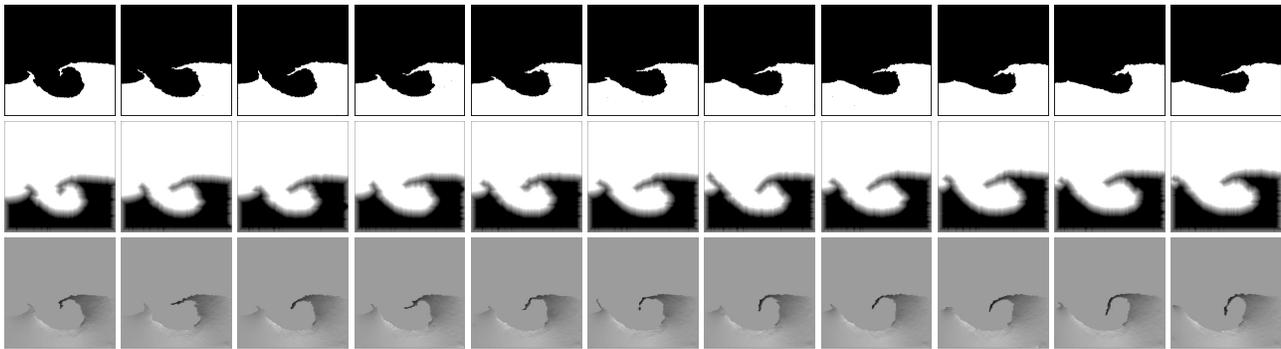


(b) Normal training data samples with noise: larger output changes but ordering still applies

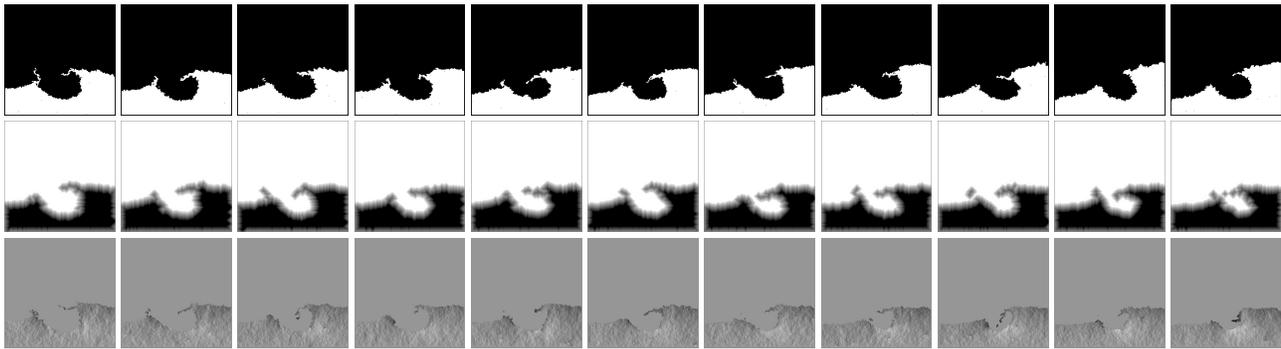


(c) Outlier data samples: noise can override the generation ordering by chance

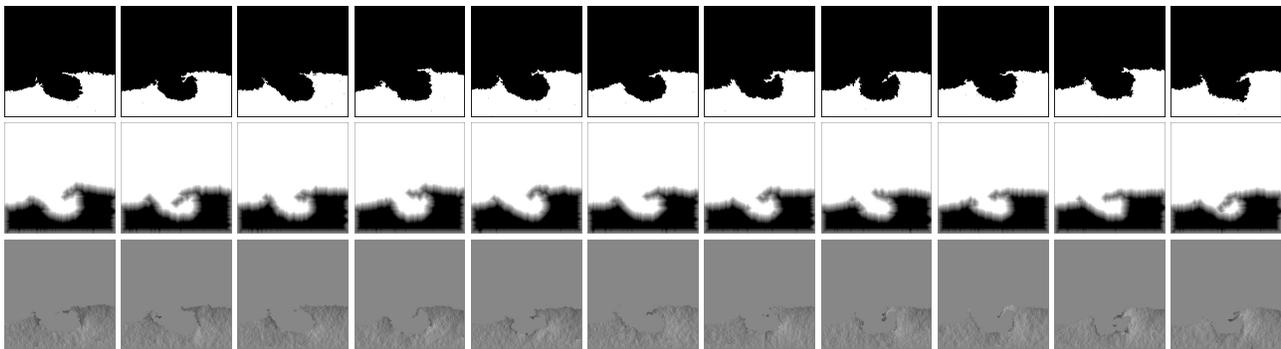
Figure 16. Various smoke simulation examples using one component of the velocity (top rows), the density (middle rows), and the pressure field (bottom rows).



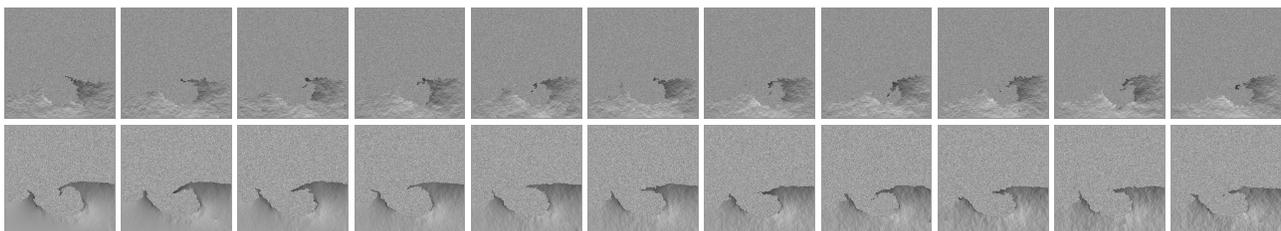
(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies

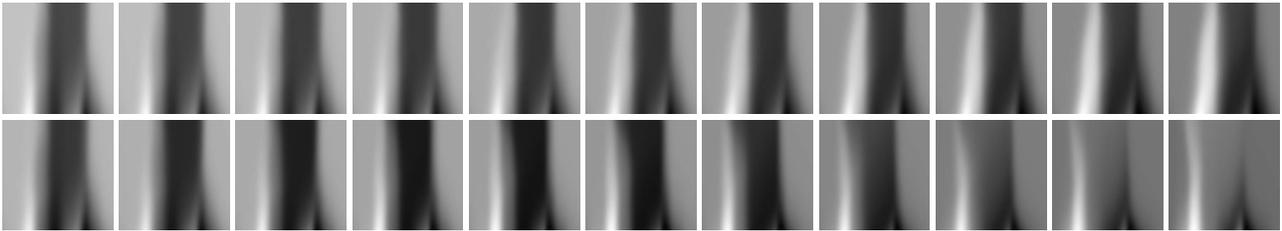


(c) Outlier data samples: noise can override the generation ordering by chance

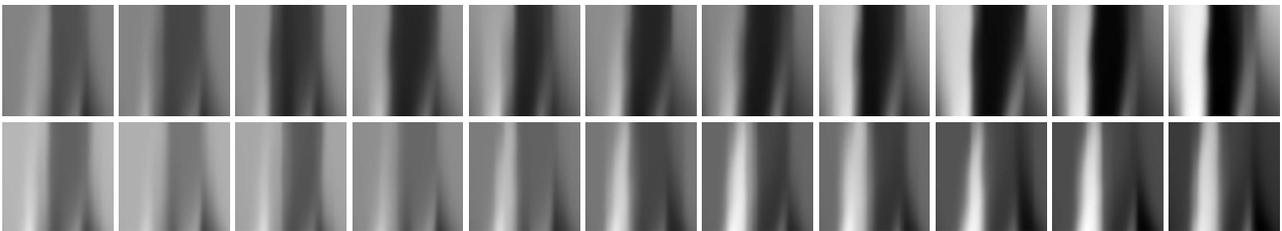


(d) Data samples from test set with additional background noise

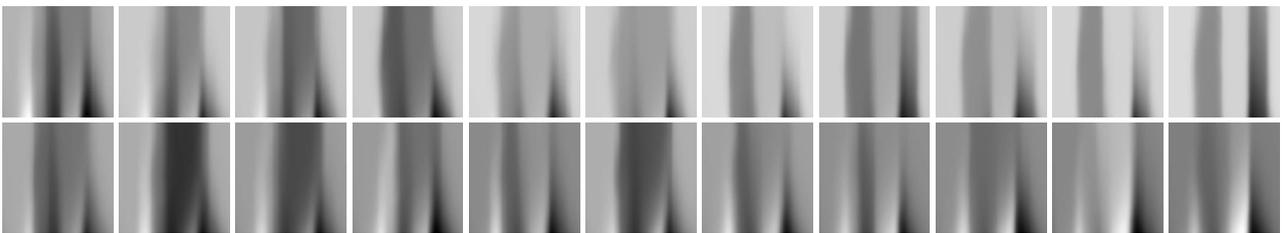
Figure 17. Several liquid simulation examples using the binary indicator flags (top rows), the extrapolated level set values (middle rows), and one component of the velocity field (bottom rows) for the training data and only the velocity field for the test data.



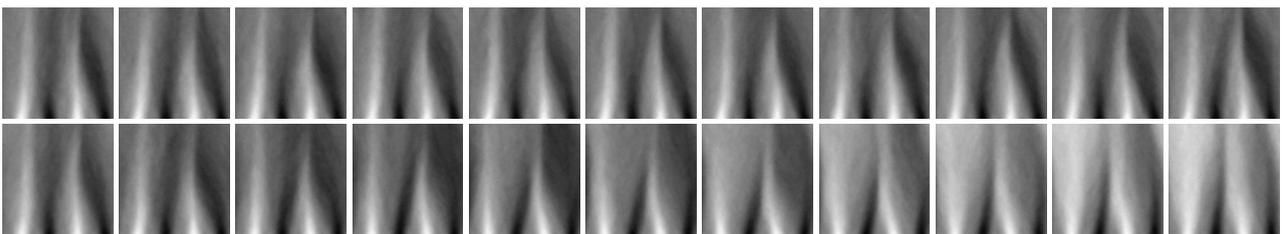
(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies

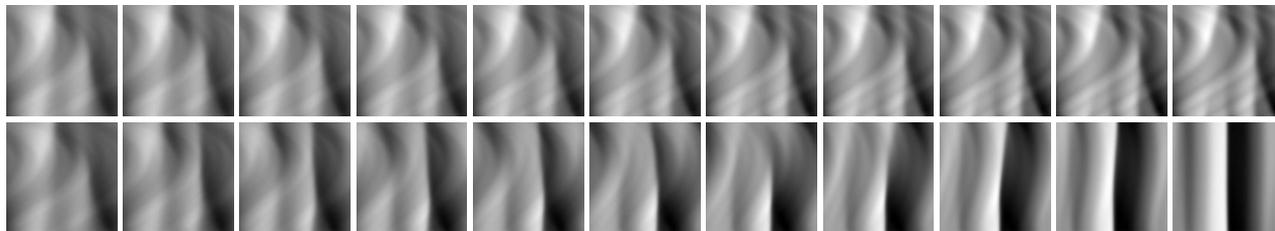


(c) Outlier data samples: noise can override the generation ordering by chance

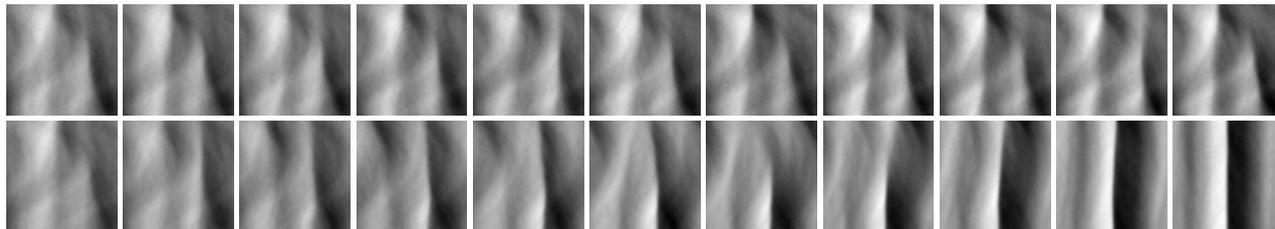


(d) Data samples from test set with additional background noise

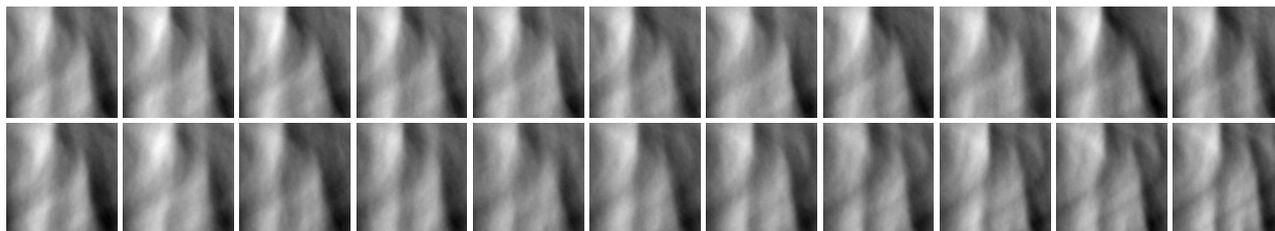
Figure 18. Various examples from the Advection-Diffusion equation using the density field.



(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies



(c) Outlier data samples: noise can override the generation ordering by chance

Figure 19. Different simulation examples from the Burger’s equation using the velocity field.

The liquid training set consists of 792 sequences and the validation set of 198 sequences with different random seeds. For the liquid test set, additional background noise was added to the velocity field of the simulations as displayed in Fig. 17(d). Because this only alters the velocity field, the extrapolated level set values and binary indicator flags are not used for this data set, leading to 132 sequences.

D.2. Advection-Diffusion and Burger’s Equation

For these PDEs, our solvers only discretize and solve the corresponding equation in 1D. Afterwards, the different time steps of the solution process are concatenated along a new dimension to form 2D data with one spatial and one time dimension.

ADVECTION-DIFFUSION EQUATION

This equation describes how a passive quantity is transported inside a velocity field due to the processes of advection and diffusion. Eq. (12) is the simplified Advection-Diffusion equation with constant diffusivity and no sources or sinks.

$$\frac{\partial d}{\partial t} = \nu \nabla^2 d - u \cdot \nabla d, \quad (12)$$

where d denotes the density, u is the velocity, and ν is the kinematic viscosity (also known as diffusion coefficient) that determines the strength of the diffusion. Our solver employed a simple implicit time integration and a diffusion solver based on conjugate gradient without preconditioning. The initialization for the 1D fields of the simulations was created by overlaying multiple parameterized sine curves with random frequencies and magnitudes.

In addition, continuous forcing controlled by further parameterized sine curves was included in the simulations over time. In this case, the only initial conditions to vary are the forcing and initialization parameters of the sine curves and the strength of the added noise. From this PDE, only the passive density field was used as shown in Fig. 18. Overall, 798 sequences are included in the training set and 190 sequences with a different random initialization in the validation set.

For the Advection-Diffusion test set, the noise was instead added directly to the passive density field of the simulations. This results in 190 sequences with more small scale details as shown in Fig. 18(d).

BURGER'S EQUATION

This equation is very similar to the Advection-Diffusion equation and describes how the velocity field itself changes due to diffusion and advection:

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - u \cdot \nabla u. \quad (13)$$

Eq. (13) is known as the viscous form of the Burger's equation that can develop shock waves, and again u is the velocity and ν denotes the kinematic viscosity. Our solver for this PDE used a slightly different implicit time integration scheme, but the same diffusion solver as used for the Advection-Diffusion equation.

The simulation setup and parameters were also the same; the only difference is that the velocity field instead of the density is exported. As a consequence, the data in Fig. 19 looks relatively similar to those from the Advection-Diffusion equation. The training set features 782 sequences, and the validation set contains 204 sequences with different random seeds.

D.3. Other Data-Sets

The remaining data sets are not based on PDEs and thus not generated with the proposed method. The data is only used to test the generalization of the discussed metrics and not for training or validation. The Shapes test set contains 160 sequences, the Video test set consists 131 sequences, and the TID test set features 216 sequences.

SHAPES

This data set tests if the metrics are able to track simple, moving geometric shapes. To create it, a straight path between two random points inside the domain is generated

and a random shape is moved along this path in steps of equal distance. The size of the used shape depends on the distance between the start and end point such that a significant fraction of the shape overlaps between two consecutive steps. It is also ensured that no part of the shape leaves the domain at any step by using a sufficiently big boundary area when generating the path.

With this method, multiple random shapes for a single data sample are produced, and their paths can overlap such that they occlude each other to provide an additional challenge. All shapes are moved in their parametric representation, and only when exporting the data, they are discretized onto a fixed binary grid. To add more variations to this simple approach, we also apply them in a non-binary way with smoothed edges and include additive Gaussian noise over the entire domain. Examples are shown in Fig. 20.

VIDEO

For this data set, different publicly available video recordings were acquired and processed in three steps. First, videos with abrupt cuts, scene transitions, or camera movements were discarded, and afterwards the footage was broken down into single frames. Then, each frame was resized to match the spatial size of our other data by linear interpolation. Since directly using consecutive frames is no challenge for any analyzed metric and all of them recovered the ordering almost perfectly, we achieved a more meaningful data set by skipping several intermediate frames. For the final data set, we defined the first frame of every video as the reference and collected subsequent frames in an interval step of ten frames as the increasingly different variations. Some data examples can be found in Fig. 21.

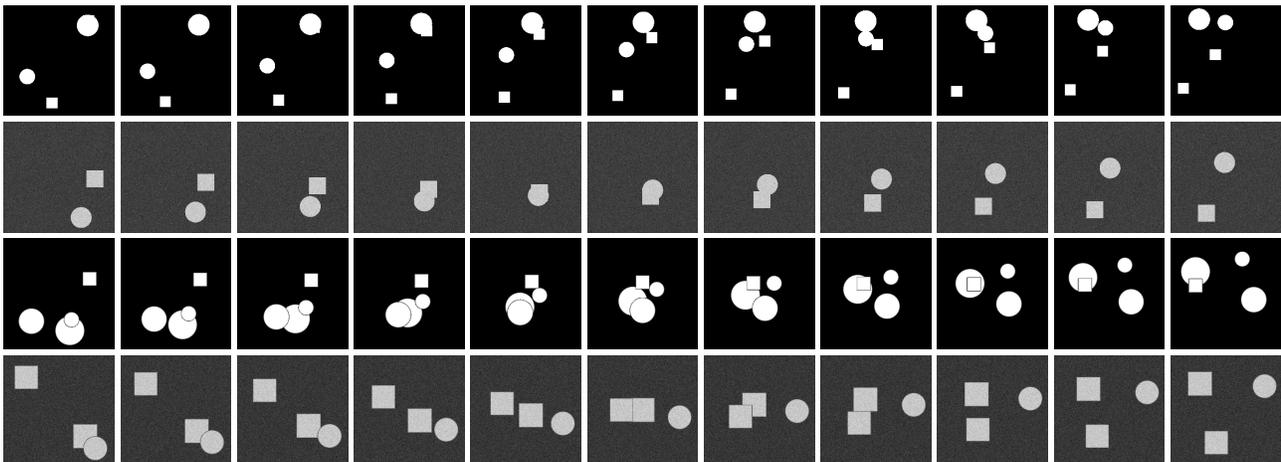


Figure 20. Examples from the shapes data set using a field with only binary shape values (first row), shape values with additional noise (second row), smoothed shape values (third row), and smoothed values with additional noise (fourth row).



Figure 21. Multiple examples from the video data set.

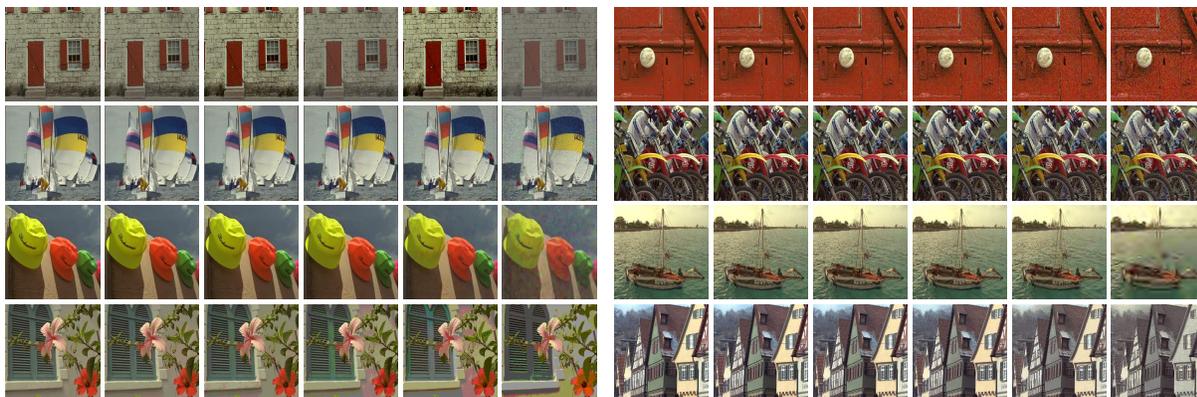


Figure 22. Examples from the TID2013 data set proposed by Ponomarenko et al.. Displayed are a change of contrast, three types of noise, denoising, jpg2000 compression, and two color quantizations (from left to right and top to bottom).

TID2013

This data set was created by Ponomarenko et al. and used without any further modifications. It consists of 25 reference images with 24 distortion types in five levels. As a result, it is not directly comparable to our data sets; thus, it is excluded from the test set aggregations. The distortions focus on various types of noise, image compression, and color changes. Fig. 22 contains examples from the data set.

D.4. Hardware

Data generation, training, and metric evaluations were performed on a machine with an Intel i7-6850 (3.60Ghz) CPU and an NVIDIA GeForce GTX 1080 Ti GPU.

E. Real-World Data

Below, we give details of the three data sets used for the evaluation in Section 6.3 of the main paper.

E.1. ScalarFlow

The *ScalarFlow* data set (Eckert et al., 2019) contains 3D velocities of real-world scalar transport flows reconstructed from multiple camera perspectives. For our evaluation, we cropped the volumetric $100 \times 178 \times 100$ grids to $100 \times 160 \times 100$ such that they only contain the area of interest and convert them to 2D with two variants: either by using the center slice or by computing the mean along the z-dimension. Afterwards, the velocity vectors are split by channels, linearly interpolated to 256×256 , and then normalized. Variations for each reconstructed plume are acquired by using frames in equal temporal intervals. We employed the velocity field reconstructions from 30 plumes (with simulation IDs 0 – 29) for both compression methods. Fig. 23 shows some example sequences.

E.2. Johns Hopkins Turbulence Database

The Johns Hopkins Turbulence Database (*JHTDB*) (Perlman et al., 2007) features various data sets of 3D turbu-

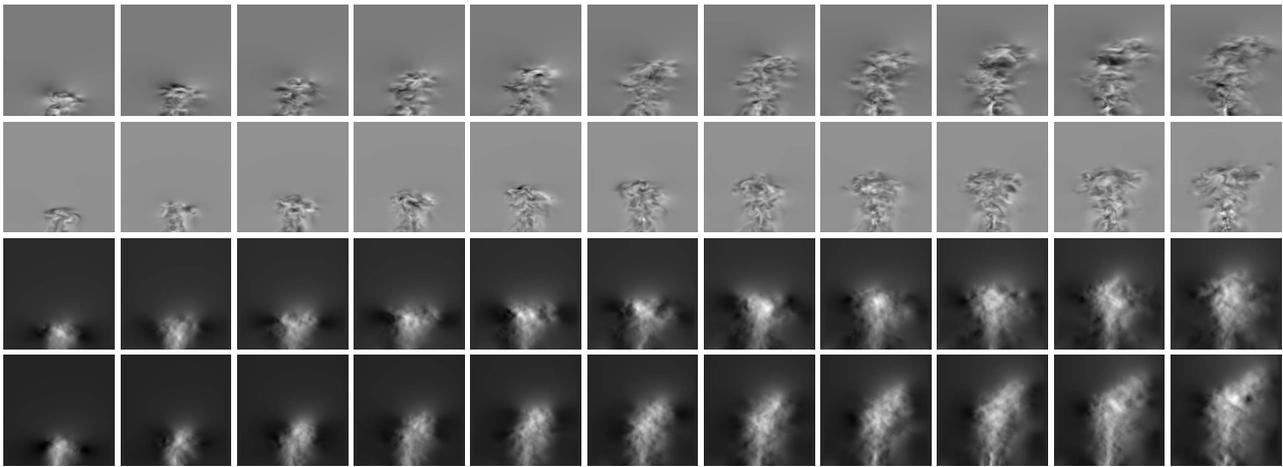


Figure 23. Four different smoke plume examples of the processed *ScalarFlow* data set using one of the three velocity components. The two top rows show the center slice, and the two bottom rows show the mean along the *z*-dimension. The temporal interval between each image is ten simulation time steps.

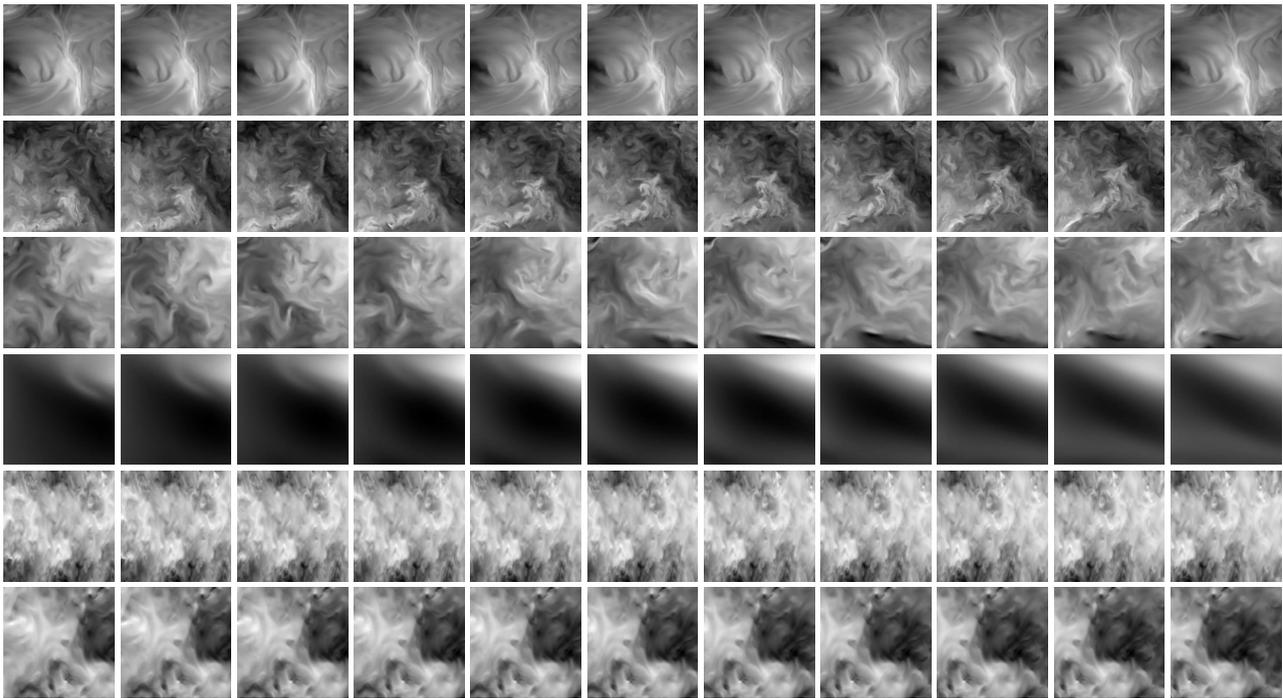


Figure 24. Data samples extracted from the Johns Hopkins Turbulence Database with a spatial or temporal interval of ten using one of the three velocity components. From top to bottom: *mhd1024* and *isotropic1024coarse* (varied time step), *isotropic4096* and *rotstrat4096* (varied *z*-position), *channel* and *channel5200* (varied *x*-position).

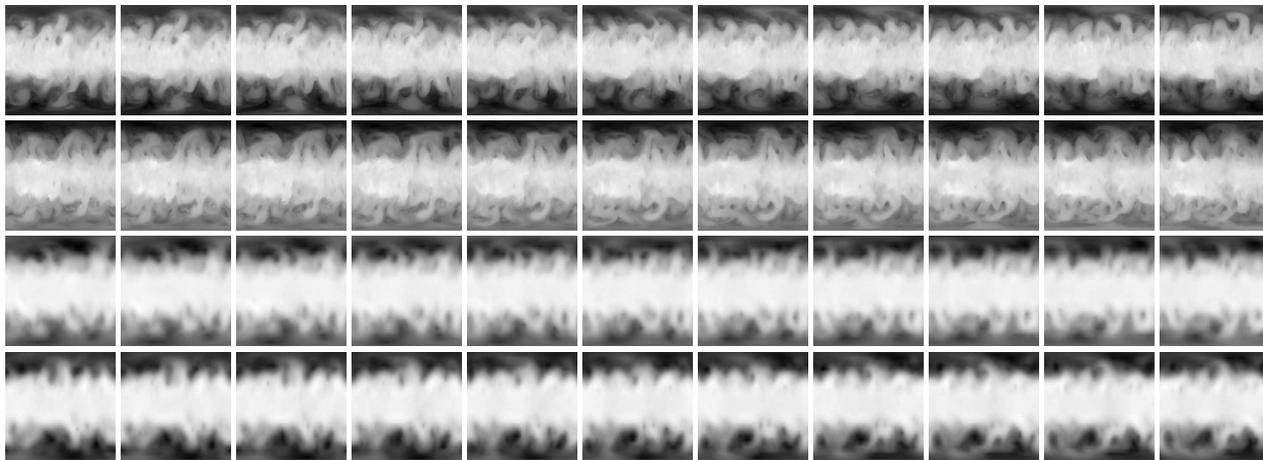


Figure 25. Examples of the processed *WeatherBench* data: high-res temperature data $1.40625deg/temperature$ (upper two rows) and low-res geopotential data $5.625deg/geopotential_500$ (lower two rows). The temporal interval spacing between the images is twenty hours.

lent flow fields created with direct numerical simulations (DNS). Here, we used three forced isotropic turbulence data sets with different resolutions (*isotropic1024coarse*, *isotropic1024fine*, and *isotropic4096*), two channel flows with different Reynolds numbers (*channel* and *channel-5200*), the forced magneto-hydrodynamic isotropic turbulence data set (*mhd1024*), and the rotating stratified turbulence data set (*rotstrat4096*).

For the evaluation, five 256×256 reference slices in the x/y -plane from each of the seven data sets are used. The spatial and temporal position of each slice is randomized within the bounds of the corresponding simulation domain. We normalize the value range and split the velocity vectors by component for an individual evaluation. Variants for each reference are created by gradually varying the x - and z -position of the slice in equal intervals. The temporal position of each slice is varied as well if a sufficient amount of temporally resolved data is available (for *isotropic1024coarse*, *isotropic1024fine*, *channel*, and *mhd1024*). This leads to 216 sequences in total. Fig. 24 shows examples from six of the *JHTDB* data sets.

E.3. WeatherBench

The *WeatherBench* repository (Rasp et al., 2020) represents a collection of various weather measurements of different atmospheric quantities such as precipitation, cloud coverage, wind velocities, geopotential, and temperature. The data ranges from 1979 to 2018 with a fine temporal resolution and is stored on a Cartesian latitude-longitude grid of the earth. In certain subsets of the data, an additional dimension such as altitude or pressure levels is available. As all measurements are available as scalar fields, only a linear interpolation to the correct input size and a normalization was necessary in order to prepare the data. We used the low-

resolution geopotential data set at 500hPa (i.e., at around 5.5km height) with a size of 32×64 yielding smoothly changing features when upsampling the data. In addition, the high-res temperature data with a size of 128×256 for small scale details was used. For the temperature field, we used the middle atmospheric pressure level at 850hPa corresponding to an altitude of 1.5km in our experiments.

To create sequences with variations for a single time step of the weather data, we used frames in equal time intervals, similar to the *ScalarFlow* data. Due to the very fine temporal discretization of the data, we only use a temporal interval of two hours as the smallest interval step of one in Fig. 26. We sampled three random starting points in time from each of the 40 years of measurements, resulting in 120 temperature and geopotential sequences overall. Fig. 25 shows a collection of example sequences.

E.4. Detailed Results

For each of the variants explained in the previous sections, we create test sets with six different spatial and temporal intervals. Fig. 26 shows the combined Spearman correlation of the sequences for different interval spacings when evaluating various metrics. For the results in Fig. 7 in the main paper, all correlation values shown here are aggregated by data source via mean and standard deviation.

While our metric reliably recovers the increasing distances within the data sets, the individual measurements exhibit interesting differences in terms of their behavior for varying distances. As *JHTDB* and *WeatherBench* contain relatively uniform phenomena, a larger step interval creates more difficult data as the simulated and measured states contain changes that are more and more difficult to analyze along a sequence. For *ScalarFlow*, on the other hand, the diffi-

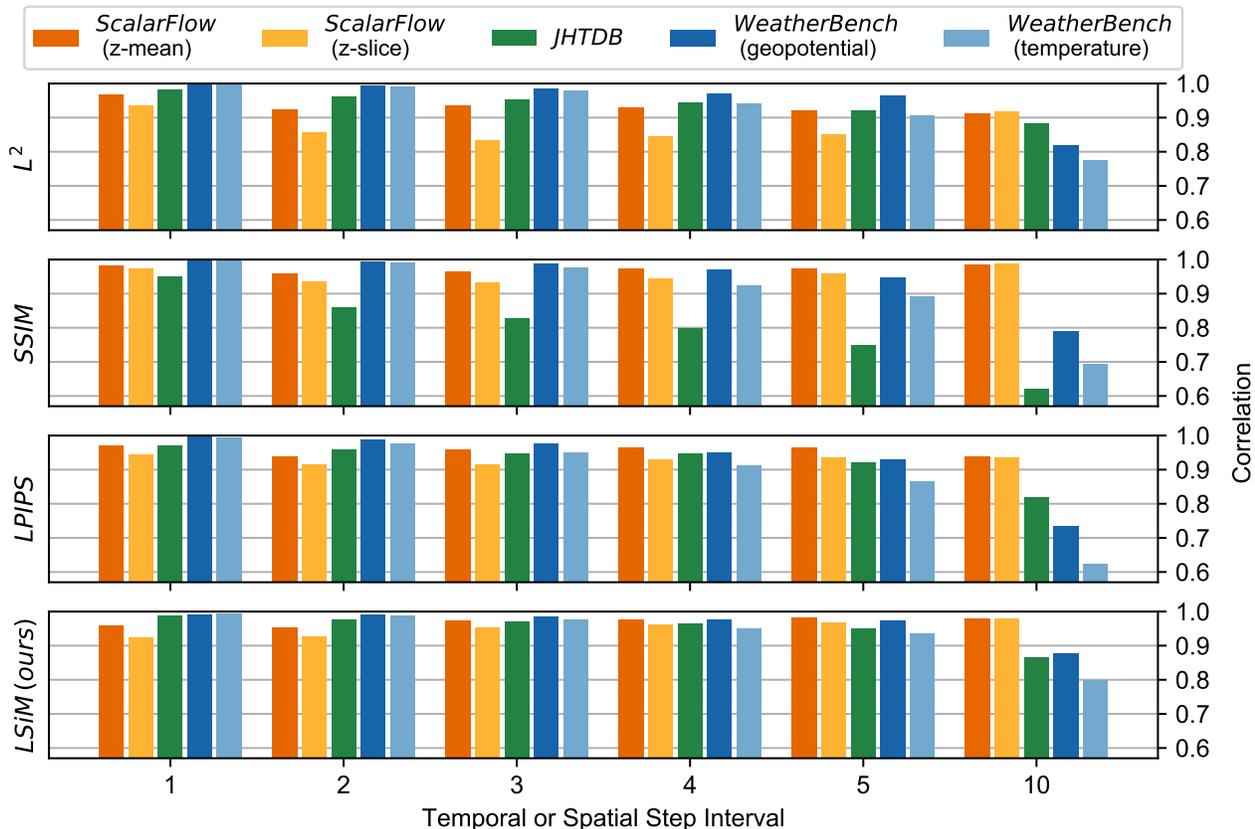


Figure 26. Detailed breakdown of the results when evaluating $LSiM$ on the individual data sets of *ScalarFlow* (30 sequences each), *JHTDB* (90 sequences each), and *WeatherBench* (120 sequences each) with different step intervals.

culty decreases for larger intervals due to the large-scale motion of the reconstructed plumes. As a result of buoyancy forces, the observed smoke rises upwards into areas where no smoke has been before. For the network, this makes predictions relatively easy as the large-scale translations are indicative of the temporal progression, and small scale turbulence effects can be largely ignored. For this data set, smaller intervals are more difficult as the overall shape of the plume barely changes while the complex evolution of small scale features becomes more important.

Overall, the $LSiM$ metric recovers the ground truth ordering of the sequences very well as indicated by the consistently high correlation values in Fig. 26. The other metrics come close to these results on certain sub-datasets but are significantly less consistent. $SSiM$ struggles on *JHTDB* across all interval sizes, and $LPIPS$ cannot keep up on *WeatherBench*, especially for larger intervals. L^2 is more stable overall, but consistently stays below the correlation achieved by $LSiM$.

F. Additional Evaluations

In the following, we demonstrate other ways to compare the performance of the analyzed metrics on our data sets. In

Tab. 2, the Pearson correlation coefficient is used instead of Spearman’s rank correlation coefficient. While Spearman’s correlation measures monotonic relationships by using ranking variables, it directly measures linear relationships.

The results in Tab. 2 match very closely to the values computed with Spearman’s rank correlation coefficient. The best performing metrics in both tables are identical; only the numbers slightly vary. Since a linear and a monotonic relation describes the results of the metrics similarly well, there are no apparent non-linear dependencies that cannot be captured using the Pearson correlation.

In the Tables 3 and 4, we employ a different, more intuitive approach to determine combined correlation values for each data set using the Pearson correlation. We are no longer analyzing the entire predicted distance distribution and the ground truth distribution at once as done above. Instead, we individually compute the correlation between the ground truth and the predicted distances for the single data samples of the data set. From the single correlation values, we compute the mean and standard deviations shown in the tables. Note that this approach potentially produces less accurate comparison results, as small errors in the individual

Table 2. Performance comparison on validation and test data sets measured in terms of the Pearson correlation coefficient of ground truth against predicted distances. **Bold+underlined** values show the best performing metric for each data set, **bold** values are within a 0.01 error margin of the best performing, and *italic* values are 0.2 or more below the best performing. On the right a visualization of the combined test data results is shown for selected models.

Metric	Validation data sets				Test data sets					
	Smo	Liq	Adv	Bur	TID	LiqN	AdvD	Sha	Vid	All
L^2	0.66	0.80	0.72	0.60	0.82	0.73	0.55	<i>0.66</i>	0.79	0.60
SSIM	0.69	0.74	0.76	0.70	0.78	<i>0.26</i>	0.69	<i>0.49</i>	0.73	0.53
LPIPS v0.1	0.63	0.68	0.66	0.71	0.85	<i>0.49</i>	0.61	0.84	0.83	0.65
AlexNet _{random}	0.63	0.69	0.67	0.65	0.83	0.64	0.63	0.74	0.81	0.65
AlexNet _{frozen}	0.66	0.69	0.68	0.71	0.85	<i>0.39</i>	0.61	0.86	0.83	0.64
Optical flow	0.63	<i>0.56</i>	<i>0.37</i>	<i>0.39</i>	<i>0.49</i>	<i>0.45</i>	<i>0.28</i>	<i>0.61</i>	0.74	<i>0.48</i>
Non-Siamese	0.77	0.84	0.78	0.74	0.67	0.81	0.64	<i>0.27</i>	0.79	0.60
Skip _{from scratch}	0.79	0.83	0.80	0.73	0.85	0.78	0.61	0.79	0.84	0.71
LSiM _{noiseless}	0.77	0.77	0.76	0.72	0.86	0.62	0.58	0.84	0.83	0.68
LSiM _{strong noise}	0.65	<i>0.64</i>	0.66	0.68	0.81	<i>0.39</i>	0.53	0.90	0.82	0.64
LSiM (ours)	0.78	0.82	0.79	0.74	0.86	0.79	0.58	0.87	0.82	0.72

Table 3. Performance comparison on validation data sets measured by computing mean and standard deviation (in brackets) of Pearson correlation coefficients (ground truth against predicted distances) from individual data samples. **Bold+underlined** values show the best performing metric for each data set, **bold** values are within a 0.01 error margin of the best performing, and *italic* values are 0.2 or more below the best performing. On the right a visualization of the combined test data results is shown for selected models.

Metric	Validation data sets			
	Smo	Liq	Adv	Bur
L^2	0.68 (0.27)	0.82 (0.18)	0.74 (0.24)	0.63 (0.33)
SSIM	0.71 (0.23)	0.75 (0.23)	0.79 (0.21)	0.73 (0.33)
LPIPS v0.1	0.66 (0.29)	0.71 (0.24)	0.70 (0.29)	0.75 (0.28)
AlexNet _{random}	0.65 (0.28)	0.71 (0.29)	0.71 (0.27)	0.68 (0.31)
AlexNet _{frozen}	0.69 (0.27)	0.72 (0.25)	0.71 (0.27)	0.74 (0.29)
Optical flow	0.66 (0.38)	<i>0.59 (0.47)</i>	<i>0.38 (0.52)</i>	<i>0.41 (0.49)</i>
Non-Siamese	0.80 (0.19)	0.87 (0.14)	0.81 (0.20)	0.76 (0.32)
Skip _{from scratch}	0.81 (0.19)	0.85 (0.16)	0.82 (0.19)	0.77 (0.30)
LSiM _{noiseless}	0.79 (0.21)	0.79 (0.20)	0.79 (0.23)	0.76 (0.29)
LSiM _{strong noise}	0.67 (0.28)	<i>0.66 (0.29)</i>	0.68 (0.30)	0.70 (0.32)
LSiM (ours)	0.81 (0.20)	0.84 (0.16)	0.81 (0.19)	0.78 (0.28)

Table 4. Performance comparison on test data sets measured by computing mean and std. dev. (in brackets) of Pearson correlation coefficients (ground truth against predicted distances) from individual data samples. **Bold+underlined** values show the best performing metric for each data set, **bold** values are within a 0.01 error margin of the best performing, and *italic* values are 0.2 or more below the best performing.

Metric	Test data sets					
	TID	LiqN	AdvD	Sha	Vid	All
L^2	0.84 (0.08)	0.75 (0.18)	0.57 (0.38)	<i>0.67 (0.18)</i>	0.84 (0.27)	0.69 (0.29)
SSIM	0.81 (0.20)	<i>0.26 (0.38)</i>	0.71 (0.31)	<i>0.53 (0.32)</i>	0.77 (0.28)	0.58 (0.38)
LPIPS v0.1	0.87 (0.11)	<i>0.51 (0.34)</i>	0.63 (0.34)	0.85 (0.14)	0.87 (0.22)	0.71 (0.31)
AlexNet _{random}	0.84 (0.10)	0.67 (0.24)	0.65 (0.33)	0.74 (0.18)	0.85 (0.26)	0.72 (0.28)
AlexNet _{frozen}	0.86 (0.11)	<i>0.41 (0.37)</i>	0.64 (0.34)	0.87 (0.14)	0.87 (0.22)	0.70 (0.34)
Optical flow	0.74 (0.67)	<i>0.50 (0.34)</i>	<i>0.32 (0.53)</i>	<i>0.63 (0.45)</i>	0.78 (0.45)	<i>0.53 (0.49)</i>
Non-Siamese	0.87 (0.12)	0.84 (0.12)	0.66 (0.34)	<i>0.31 (0.45)</i>	0.83 (0.26)	0.64 (0.39)
Skip _{from scratch}	0.87 (0.12)	0.80 (0.16)	0.63 (0.37)	0.80 (0.17)	0.87 (0.20)	0.76 (0.27)
LSiM _{noiseless}	0.87 (0.11)	<i>0.64 (0.29)</i>	0.60 (0.38)	0.86 (0.15)	0.86 (0.22)	0.73 (0.31)
LSiM _{strong noise}	0.83 (0.12)	<i>0.39 (0.38)</i>	0.55 (0.36)	0.91 (0.17)	0.86 (0.25)	0.67 (0.37)
LSiM (ours)	0.88 (0.10)	0.81 (0.15)	0.60 (0.37)	0.88 (0.16)	0.85 (0.23)	0.77 (0.28)

computations can accumulate to larger deviations in mean and standard deviation. Still, both tables lead to very similar conclusions: The best performing metrics are almost the same, and low combined correlation values match with results that have a high standard deviation and a low mean.

Fig. 27 shows a visualization of predicted distances c against ground truth distances d for different metrics on every sample from the test sets. Each plot contains over 6700 individual data points to illustrate the global distance distributions created by the metrics, without focusing on single cases. A theoretical optimal metric would recover a perfectly narrow distribution along the line $c = d$, while worse metrics recover broader, more curved distributions. Overall, the sample distribution of an L^2 metric is very wide. $LPIPS$ manages to follow the optimal diagonal a lot better, but our approach approximates it with the smallest deviations, as also shown in the tables above. The L^2 metric performs very poorly on the shape data indicated by the too steeply increasing blue lines that flatten after a ground truth distance of 0.3. $LPIPS$ already significantly reduces this problem, but $LSiM$ still works slightly better.

A similar issue is visible for the Advection-Diffusion data, where for L^2 a larger number of red samples is below the

optimal $c = d$ line, than for the other metrics. $LPIPS$ has the worst overall performance for liquid test set, indicated by the large number of fairly chaotic green lines in the plot. On the video data, all three metrics perform similarly well.

A fine-grained distance evaluation in 200 steps of L^2 and our $LSiM$ metric via the mean and standard deviation of different data samples is shown in Fig. 28. Similar to Fig. 27, the mean of an optimal metric would follow the ground truth line with a standard deviation of zero, while the mean of worse metrics deviates around the line with a high standard deviation. The plot on the left combines eight samples with different seeds from the *Sha* data set, where only a single shape is used. Similarly, the center plot aggregates eight samples from *Sha* with more than one shape. The right plot shows six data samples from the *LiqN* test set that vary by the amount of noise that was injected into the simulation.

The task of only tracking a single shape in the example on the left is the easiest of the three shown cases. Both metrics have no problem to recover the position change until a variation of 0.4, where L^2 can no longer distinguish between the different samples. Our metric recovers distances with a continuously rising mean and a very low standard deviation. The task in the middle is already harder, as multiple shapes

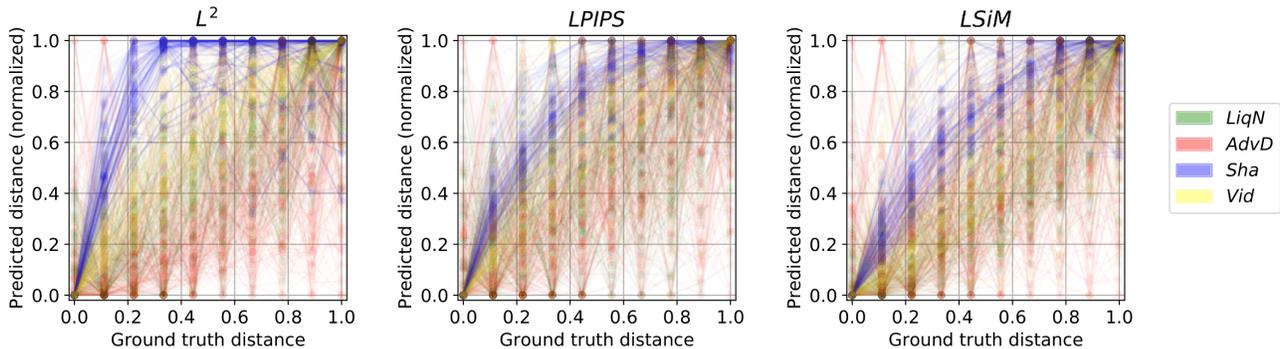


Figure 27. Distribution evaluation of ground truth distances against normalized predicted distances for L^2 , $LPIPS$ and $LSiM$ on all test data (color coded).

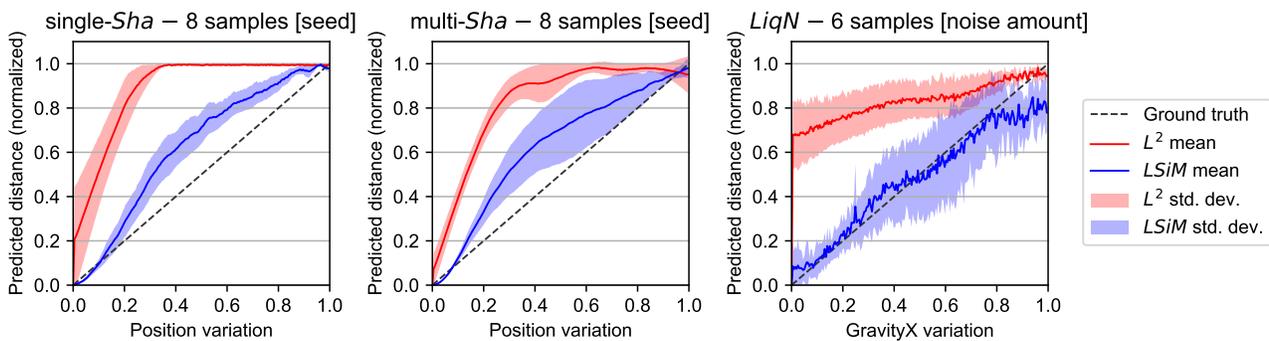


Figure 28. Mean and standard deviation of normalized distances over multiple data samples for L^2 and $LSiM$. The samples differ by the quantity displayed in brackets. Each data sample uses 200 parameter variation steps instead of 10 like the others in our data sets. For the shape data the position of the shape varies and for the liquid data the gravity in x-direction is adjusted.

can occlude each other during the position changes. Starting at a position variation of 0.4, both metrics have a quite high standard deviation, but the proposed method stays closer to the ground truth line. L^2 shows a similar issue as before because it flattens relatively fast. The plot on the right features the hardest task. Here, both metrics perform similar as each has a different problem in addition to an unstable mean. Our metric stays close to the ground truth, but has a quite high standard deviation starting at about a variation of 0.4. The standard deviation of L^2 is lower, but instead it starts off with a big jump from the first few data points. To some degree, this is caused by the normalization of the plots, but it still overestimates the relative distances for small variations in the simulation parameter.

These findings also match with the distance distribution evaluations in Fig. 27 and the tables above: Our method has a significant advantage over shallow metrics on shape data, while the differences of both metrics become much smaller for the liquid test set.

G. Notation

In this work, we follow the notation suggested by Goodfellow et al.. Vector quantities are displayed in bold, and tensors use a sans-serif font. Double-barred letters indicate sets or vector spaces. The following symbols are used:

\mathbb{R}	Real numbers	o_0, o_1, \dots, o_n	Series of outputs of a simulation with increasing ground truth distance to o_0
i, j	Indexing in different contexts	Δ	Amount of change in a single simulation parameter
\mathbb{I}	Input space of the metric, i.e., color images/field data of size $224 \times 224 \times 3$	t_1, t_2, \dots, t_t	Time steps of a numerical simulation
a	Dimension of the input space \mathbb{I} when flattened to a single vector	v	Variance of the noise added to a simulation
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Elements in the input space \mathbb{I}	\mathbf{c}	Ground truth distance distribution, determined by the data generation via Δ
\mathbb{L}	Latent space of the metric, i.e., sets of 3 rd order feature map tensors	\mathbf{d}	Predicted distance distribution (supposed to match the corresponding \mathbf{c})
b	Dimension of the latent space \mathbb{L} when flattened to a single vector	$\bar{\mathbf{c}}, \bar{\mathbf{d}}$	Mean of the distributions \mathbf{c} and \mathbf{d}
$\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}$	Elements in the latent space \mathbb{L} , corresponding to $\mathbf{x}, \mathbf{y}, \mathbf{z}$	$\ \cdot\cdot\cdot\ _2$	Euclidean norm of a vector
w	Weights for the learned average aggregation (1 per feature map)	$m(\mathbf{x}, \mathbf{y})$	Entire function computed by our metric
p_0, p_1, \dots	Initial conditions / parameters of a numerical simulation	$m_1(\mathbf{x}, \mathbf{y})$	First part of $m(\mathbf{x}, \mathbf{y})$, i.e., base network and feature map normalization
n	Number of variations of a simulation parameter, thus determines length of the network input sequence	$m_2(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$	Second part of $m(\mathbf{x}, \mathbf{y})$, i.e., latent space difference and the aggregations
		\mathbf{G}	3 rd order feature tensor from one layer of the base network
		g_b, g_c, g_x, g_y	Batch (g_b), channel (g_c), and spatial dimensions (g_x, g_y) of \mathbf{G}
		f	Optical flow network
		f^{xy}, f^{yx}	Flow fields computed by an optical flow network f from two inputs in \mathbb{I}
		f_1^{xy}, f_2^{xy}	Components of the flow field f^{xy}
		∇, ∇^2	Gradient (∇) and Laplace operator (∇^2)
		∂	Partial derivative operator
		t	Time in our PDEs
		u	Velocity in our PDEs
		ν	Kinematic viscosity / diffusion coefficient in our PDEs
		d, ρ	Density in our PDEs
		P	Pressure in the Navier-Stokes Equations
		g	Gravity in the Navier-Stokes Equations