Learning Similarity Metrics for Volumetric Simulations with Multiscale CNNs

Georg Kohl
Department of Informatics
Technical University of Munich
Munich, Germany
`georg.kohl@tum.de`

Li-Wei Chen
Department of Informatics
Technical University of Munich
Munich, Germany
`liwei.chen`

Nils Thuerey
Department of Informatics
Technical University of Munich
Munich, Germany
`nils.thuerey@tum.de`

Abstract

We propose a similarity model based on entropy, which allows for the creation of physically meaningful ground truth distances for the similarity assessment of scalar and vectorial data, produced from transport and motion-based simulations. Utilizing two data acquisition methods derived from this model, we create collections of fields from numerical PDE solvers and existing simulation data repositories. Furthermore, a multiscale CNN architecture that computes a volumetric similarity metric (*VolSiM*) is proposed and its robustness is evaluated on a large range of test data. To the best of our knowledge this is the first learning method inherently designed to address the similarity assessment of high-dimensional simulation data.

1 Introduction and Related Work

Making comparisons is a fundamental operation for the simulation of physical phenomena, as comparisons against other models or measurements frequently occur. A recurring problem is that traditional comparisons are typically based on simple, element-wise metrics like the L^n distances, due to their computational simplicity and a lack of alternatives. Physical problems often exhibit strong dependencies between elements in their solutions that should be considered, but are by definition ignored by such metrics. This is especially problematic for systems that are modeled with dense grid data, as the number of interactions grows exponentially with a linearly increasing number elements. Such data representations are widely used, e.g. for medical blood flow simulations [Olufsen et al., 2000], climate and weather predictions [Stocker et al., 2014], and even the famous unsolved problem of turbulence [Holmes et al., 2012]. Another downside of element-wise metrics is that each element is weighted equally, which is typically suboptimal; e.g. smoke plumes behave differently along the vertical dimension due to gravity or buoyancy, and small key features like vortices are more indicative of the fluid's general behavior than large areas of near constant flow [Pope, 2000].

Two metrics commonly used across disciplines are PSNR and SSIM from Wang et al. [2004], but both share the issues of element-wise metrics [Huynh-Thu and Ghanbari, 2008, 2012, Horé and Ziou, 2010]. Furthermore, statistical tools like the Pearson correlation PCC [Pearson, 1920] and Spearman's rank correlation SRCC [Spearman, 1904] can be employed as simple similarity measurements. Especially for images, similarity measurements have been approached in various ways, mostly by combining deep embeddings as perceptually more accurate metrics [Prashnani et al.,

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

2018, Talebi and Milanfar, 2018] for applications such as super-resolution [Johnson et al., 2016] or generative tasks [Dosovitskiy and Brox, 2016]. Similarity metrics for simulation data have not been studied extensively yet. Siamese networks for finding similar fluid descriptors have been applied to smoke flow synthesis [Chu and Thuerey, 2017], and Um et al. [2017, 2021] used crowd-sourced user studies for the similarity assessment of liquid simulations. Scalar 2D simulation data was previously compared with a learned metric [Kohl et al., 2020], but their *LSiM* method relies on a basic feature extractor CNN and does not account for the behavior of systems with respect to entropy. Overall, our work contributes the following:

- We propose a novel similarity model based on the entropy of physical systems. It is employed to synthesize sequences of volumetric physical fields suitable for metric learning.
- We show that our Siamese multiscale network results in a stable metric that successfully generalizes to new physical phenomena across a large range of volumetric test sets.

The central application of the proposed *VolSiM* metric is the similarity assessment of new physical simulation methods against a known ground truth from measurements, higher resolution simulations, or existing models. Furthermore, the trained metric can be used as a differentiable similarity loss for physical learning problems, similar to perceptual losses for computer vision tasks.

2 Modeling Similarity of Simulations

To formulate our methodology for learning similarity metrics that target dissipative physical systems, we turn to the fundamental quantity of entropy. The second law of thermodynamics states that the entropy S of a closed physical system never decreases, thus $\Delta S \geq 0$. In the following, we make the reasonable assumption that the behavior of the system is continuous and non-oscillating, and that $\Delta S > 0$.[1] The Boltzmann equation $S = k_B \log(W)$ from statistical mechanics describes S in terms of the Boltzmann constant k_b and the number of microstates W of a system [Boltzmann, 1866].[2] Since entropy only depends on a single system state, it can be reformulated to take the relative change between two states into account. From an information-theoretical perspective, this is related to using Shannon entropy [Shannon, 1948] as a diversity measure, as done by Rényi [1961]. Given a sequence of states $s_0, s_1, \dots, s_n$, we define the relative entropy $\tilde{S}(\boldsymbol{s}) = k \log(10^c \boldsymbol{w}_s)$. Here, $\boldsymbol{w}_s$ is the monotonically increasing, relative number of microstates defined as 0 for s_0 and as 1 for s_n. $10^c > 0$ is a system-dependent factor that determines how quickly the number of microstates increases, i.e. it represents the speed at which different processes decorrelate. As the properties of similarity metrics dictate that distances are always non-negative and only zero for identical states, the lower bound is adjusted to 0, leading to a first similarity model $\hat{D}(\boldsymbol{s}) = k \log(10^c\, \boldsymbol{w}_s + 1)$. Finally, relative similarities are equivalent up to a multiplicative constant and thus we can freely choose $k = 1/(\log 10^c + 1)$, leading to the full similarity model

$$\mathrm{D}(\boldsymbol{s}) = \frac{\log(10^c\, \boldsymbol{w}_s + 1)}{\log(10^c + 1)}. \tag{1}$$

For a sequence $\boldsymbol{s}$, it predicts the overall similarity behavior between the different states with respect to entropy, given the relative number of microstates $\boldsymbol{w}_s$ and the system decorrelation speed c. Fig. 1 illustrates the connection between the logarithmically increasing entropy and the proposed similarity model for a state sequence with length n. Here, Δ denotes the magnitude of change between the individual sequence states which is directly related to $\boldsymbol{w}_s$, and c is the decorrelation speed of the system that produced the sequence. For an informative pairwise similarity analysis, sequences should not decorrelate too quickly (*high difficulty*, red dotted) or too slowly (*low difficulty*, green dashed), but evenly exhibit both regimes (black curve). The central challenges now become finding sequences with a suitable magnitude of Δ, determining c, and assigning pairwise distances $\boldsymbol{d}$.

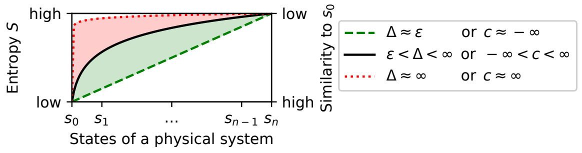


Figure 1: Idealized model of the behavior of entropy and similarity for different Δ and c.

[1]These assumptions are required to create sequences with meaningful ground truth distances in Sec. 3.

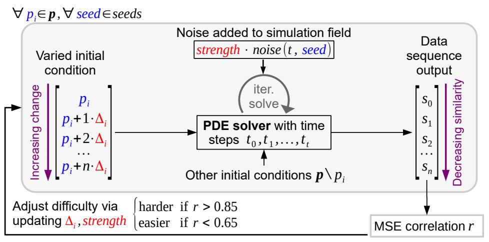
[2]We do not have any a priori information about the distribution of the likelihood of each microstate in a general physical system. Thus, the Boltzmann entropy definition which assumes a uniform microstate distribution is used in the following, instead of more generic entropy models such as the Gibbs or Shannon entropy.

3 Sequence Creation and Learning Problem Definition

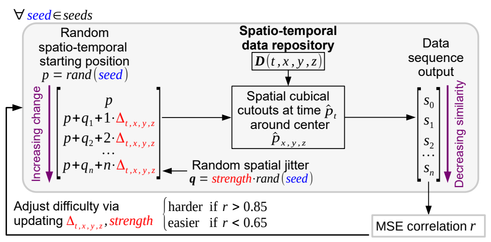
As we can neither directly determine c nor $\boldsymbol{d}$ at this point, we propose to use proxies for them during the sequence generation. Initially, this allows for finding sequences that roughly fall in a suitable difficulty range, and accurate values can be computed afterwards. Here, we use the mean squared error (MSE) as a proxy distance function and the PCC to determine c, to iteratively update Δ to a suitable range: Given any value of Δ and a corresponding sequence, pairwise proxy distances[3] between the sequence elements are computed $\boldsymbol{d}^{\Delta} = \mathrm{MSE}(s_i, s_j)$ and min-max normalized to $[0, 1]$. Next, we determine a distance sequence corresponding to the physical changes over the states, which we model as a simple linear increase over the sequence $\boldsymbol{w}_s = (j - i)/n$ following Kohl et al. [2020]. To indirectly determine c, we compare how both distance sequences differ in terms of the PCC as $r = \mathrm{PCC}(\boldsymbol{d}^{\Delta}, \boldsymbol{w}_s)$. We empirically determined that correlations between 0.65 and 0.85 work well for all cases we considered. In practice, the network stops learning effectively for $r < 0.65$ as states are too different, while sequences with $r > 0.85$ reduce generalization as a simple metric is sufficient to describe them. Using these thresholds, we propose two semi-automatic iterative methods to create data, depending on the method to introduce variations to a given state (see Fig. 2). Both methods create sequences consisting of the states $s_0, s_1, \ldots, s_n$ with decreasing similarity to the reference state s_0, where all states have the same dimensionality. Furthermore, both sample a small set of sequences to calibrate Δ to a suitable magnitude and use that value for the full data set.

[A] Variations from Initial Conditions of Simulations Given a numerical PDE solver and a set of initial conditions or parameters $\boldsymbol{p}$, the solver computes a solution to the PDE over the time steps $t_0, t_1, \ldots, t_t$. To create a larger number of different sequences, we make the systems non-deterministic by adding noise to a simulation field and randomly generating the initial conditions from a given range. Adjusting *one* of the parameters p_i in steps with a small perturbation Δ_i, allows for the creation of a sequence $s_0, s_1, \ldots, s_n$ with decreasing similarity to the unperturbed simulation output s_0. This is repeated for every suitable parameter in $\boldsymbol{p}$, and the corresponding Δ is updated individually until the targeted MSE correlation range is reached.

[B] Variations from Spatio-temporal Coherences For a source **D** of volumetric spatio-temporal data without access to a solver, we rely on a larger spatial and/or temporal dimension than the one required for a sequence. We start at a random spatio-temporal position p to extract a cubical spatial area s_0 around it. p can be repeatedly translated in space and/or time by $\Delta_{t,x,y,z}$ to create a sequence $s_0, s_1, \ldots, s_n$ of decreasing similarity. It is possible to add some global random perturbations q to the positions to further increase the difficulty.



∀ $p_i \in \boldsymbol{p}$, ∀ $seed \in seeds$
Noise added to simulation field
$strength \cdot noise(t, seed)$
Varied initial condition
Data sequence output
iter. solve
Increasing change
p_i
$p_i + 1 \cdot \Delta_i$
$p_i + 2 \cdot \Delta_i$
…
$p_i + n \cdot \Delta_i$
PDE solver with time steps $t_0, t_1, \ldots, t_t$
Other initial conditions $\boldsymbol{p} \setminus p_i$
s_0 s_1 s_2 … s_n
Decreasing similarity
Adjust difficulty via updating Δ_i, $strength$
harder if $r > 0.85$
easier if $r < 0.65$
MSE correlation r



∀ $seed \in seeds$
Random spatio-temporal starting position $p = rand(seed)$
Spatio-temporal data repository
$\boldsymbol{D}(t, x, y, z)$
Data sequence output
Increasing change
p
$p + q_1 + 1 \cdot \Delta_{t,x,y,z}$
$p + q_2 + 2 \cdot \Delta_{t,x,y,z}$
…
$p + q_n + n \cdot \Delta_{t,x,y,z}$
Spatial cubical cutouts at time $\hat{p}_t$ around center $\hat{p}_{x,y,z}$
Random spatial jitter $\boldsymbol{q} = strength \cdot rand(seed)$
s_0 s_1 s_2 … s_n
Decreasing similarity
Adjust difficulty via updating $\Delta_{t,x,y,z}$, $strength$
harder if $r > 0.85$
easier if $r < 0.65$
MSE correlation r

Figure 2: Iteration schemes to calibrate and create data sequences of decreasing similarity. Variation from the reference state can be introduced via the initial conditions of a numerical PDE simulation (method [A], left), or via spatio-temporal changes on data from a repository (method [B], right).

Data Sets To create data with method [A], we utilize solvers for a basic Advection-Diffusion model (`Adv`), Burgers' equation (`Bur`) with an additional viscosity term, and the full Navier-Stokes equations via a Eulerian smoke simulation (`Smo`) and a hybrid Eulerian-Lagrangian liquid simulation (`Liq`). The corresponding validation sets are generated with a separate set of random seeds. Furthermore, we use adjusted versions of the noise integration for two test sets, by adding noise to the density instead of the velocity in the Advection-Diffusion model (`AdvD`) and overlaying background noise in the liquid simulation (`LiqN`). We create seven test sets via method [B]. Four come from the Johns Hopkins Turbulence Database [Perlman et al., 2007] that contains a large amount of DNS data, where each is based on a subset of the JHTDB and features simulations with different characteristics (`Iso`,

[3]To keep the notation concise, sequentially indexing the distance vectors $\boldsymbol{d}^{\Delta}$ and $\boldsymbol{w}_s$ with i and j is omitted.

`Cha`, `Mhd`, and `Tra`). In addition, one test set (`SF`) via temporal translations is based on ScalarFlow [Eckert et al., 2019], consisting of 3D reconstructions of real smoke plumes. Finally, we procedurally synthesize spatial fields instead of using a data repository, to create a randomized shape (`Sha`) and damped waves (`Wav`) data set. All data was gathered in sequences with $n = 10$ at resolution 128^3, and downsampled to 64^3 for computational efficiency during training and evaluations. App. B contains further generation details as well as example visualizations for each data set.

Determining c For each calibrated sequence, we can now more accurately estimate c. As c corresponds to the decorrelation speed of the system, we choose Pearson's distance $d_i^\Delta = 1 - \text{PCC}(s_0, s_i)$ as a distance proxy here. c is determine via standard unbounded least-squares optimization from the similarity model as $c = \arg\min_c \frac{\log(10^c\, \boldsymbol{d}^\Delta + 1)}{\log(10^c + 1)}$.

Learning Setup Given the calibrated sequences $\boldsymbol{s}$ of different physical systems with elements $s_0, s_1, \dots, s_n$, the corresponding value of c, and the pairwise physical target distance sequence $\boldsymbol{w}_s = (j - i)/n$, we can now formulate a semi-supervised learning problem: We train a neural network m that receives pairs from $\boldsymbol{s}$ as an input, and outputs scalar distances $\boldsymbol{d}$ for each pair. These predictions are trained against ground truth distances $\boldsymbol{g} = \frac{\log(10^c\, \boldsymbol{w}_s + 1)}{\log(10^c + 1)}$ determined by the sequence order, transformed according to the entropy-based similarity model. Following Kohl et al. [2020], we use the loss

$$\text{L}(\boldsymbol{d}, \boldsymbol{g}) = \lambda_1 (\boldsymbol{d} - \boldsymbol{g})^2 + \lambda_2 \left(1 - \left(\sum_{i=1}^n (d_i - \bar{d})\,(g_i - \bar{g}) \middle/ \sqrt{\sum_{i=1}^n (d_i - \bar{d})^2}\, \sqrt{\sum_{i=1}^n (g_i - \bar{g})^2} \right) \right) \quad (2)$$

consisting of a weighted MSE and an inverted correlation term, where $\bar{d}$ and $\bar{g}$ denote the mean.

Network Structure For our *VolSiM* distance metric, we generally follow the established Siamese network structure, that was originally proposed for 2D domains [Zhang et al., 2018]: First, two inputs are embedded in a latent space using a CNN as a feature extractor. The Siamese structure means that the weights are shared, which ensures the mathematical requirements for a pseudo-metric [Kohl et al., 2020]. Next, the features from all layers are normalized, compared with an element-wise comparison like an absolute or squared difference, and aggregated to a scalar distance value. We propose a multiscale network to compute *VolSiM*, since physical systems often exhibit self-similar behavior that does not significantly change across scales. Generally, scaling a data pair should not alter its similarity, and networks can learn such an invariance to scale most effectively by processing data at different scales. For learned image metrics, this invariance is also useful (but less crucial), and often introduced with large strides and kernels in the convolutions, e.g. via a feature extractor based on AlexNet [Zhang et al., 2018]. However, we propose to directly encode this scale structure in a multiscale architecture for a more accurate similarity assessment, and a network with a smaller resource footprint. Fig. 3 shows proposed fully convolutional network, that effectively learns a mixture of connected deep features and similar representations across scales. Additional network and training details can be found in App. A.

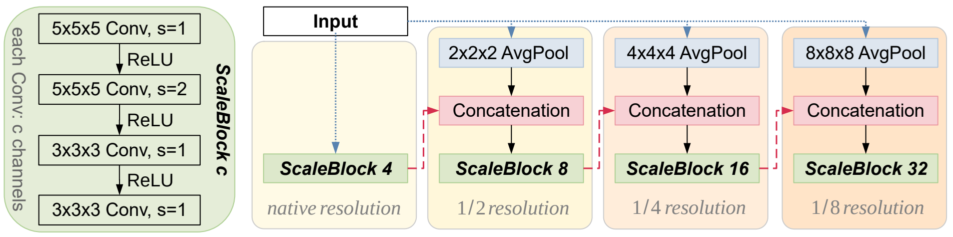


Figure 3: Conv+ReLU blocks (left) are interwoven with input and resolution connections (blue dotted and red dashed), to form the combined network architecture (right) with about $350k$ weights.

4 Results

To determine the accuracy of any metric during inference in the following, we compute the SRCC between the distance predictions of the metric $\boldsymbol{d}$ and the ground truth $\boldsymbol{w}_s$, where a value closer to 1 indicates a better reconstruction.[4] We compare different existing methods on our validation and

[4]This is equivalent to $\text{SRCC}(\boldsymbol{d}, \boldsymbol{g})$, but is computationally more efficient and has numerical benefits.

test sets at the top of Tab. 1. The proposed *VolSiM* metric consistently reconstructs the ground truth distances more reliably than other approaches on most data sets. As expected, this effect is most apparent on the validation sets since their distribution is closest to the training data. But even on the majority of test sets with a very different distribution, *VolSiM* is the best or close to the best performing metric. Metrics without deep learning such as *SSIM* [Wang et al., 2004] or variation of information *VI* [Meilă, 2007] often fall short, indicating that they were designed for different domains. The strictly element-wise metrics *MSE* and *PSNR* exhibit almost identical performance, and both work poorly on a variety of data sets. As the learning-based methods *LPIPS* [Zhang et al., 2018] and *LSiM* [Kohl et al., 2020] are limited to 2D, their assessments in Tab. 1 are obtained by averaging sliced evaluations for all three spatial axes. Both methods show improvements over the element-wise metrics, but are still clearly inferior to the performance of *VolSiM*. *LSiM* can only come close to *VolSiM* on less challenging data sets where correlation values are close to 1 and all learned reconstructions are already highly accurate. This improvement is comparable to using *LPIPS* over *PSNR*, and represents a significant step forward in terms of a robust similarity assessment.

	Validation data sets				**Test data sets**									
	Simulated				Simulated		Generated		JHTDB[a]				SF[b]	[c]
Metric	`Adv`	`Bur`	`Liq`	`Smo`	`AdvD`	`LiqN`	`Sha`	`Wav`	`Iso`	`Cha`	`Mhd`	`Tra`	`SF`	`All`
MSE	0.61	0.70	0.51	0.68	0.77	0.76	0.75	0.65	0.76	**0.86**	**0.80**	0.79	0.79	0.70
PSNR	0.61	0.68	0.52	0.68	0.78	0.76	0.75	0.65	**0.78**	**0.86**	**0.81**	0.83	0.79	0.73
SSIM	**0.75**	0.68	0.49	0.64	0.81	0.80	0.76	0.88	0.49	0.55	0.62	0.60	0.44	0.61
VI	0.57	0.69	0.43	0.60	0.69	0.82	0.67	0.87	0.59	0.76	0.68	0.67	0.41	0.62
LPIPS (2D)	0.63	0.62	0.35	0.56	0.76	0.62	0.87	0.92	0.71	0.83	0.79	0.76	0.87	0.76
LSiM (2D)	0.57	0.55	0.48	0.71	0.79	0.75	0.93	**0.97**	0.69	**0.86**	0.79	0.81	**0.98**	0.81
VolSiM (ours)	**0.75**	**0.73**	**0.66**	**0.77**	**0.84**	**0.88**	**0.95**	**0.96**	**0.77**	**0.86**	**0.81**	**0.88**	0.95	**0.85**
$CNN_{trained}$	0.60	0.71	0.63	0.76	0.81	0.77	0.92	0.93	0.75	0.86	0.78	0.85	0.95	0.82
$MS_{identity}$	0.75	0.71	0.68	0.73	0.83	0.85	0.87	0.96	0.74	0.87	0.77	0.87	0.94	0.82
$MS_{3\ scales}$	0.70	0.69	0.70	0.73	0.83	0.82	0.95	0.94	0.76	0.87	0.80	0.88	0.93	0.83
$MS_{5\ scales}$	0.78	0.72	0.78	0.78	0.81	0.90	0.94	0.93	0.75	0.85	0.77	0.88	0.93	0.82
$MS_{added\ \mathtt{Iso}}$	0.73	0.72	0.77	0.79	0.84	0.84	0.92	0.97	0.79	0.87	0.80	0.86	0.97	0.84
$MS_{only\ \mathtt{Iso}}$	0.58	0.62	0.32	0.63	0.78	0.65	0.72	0.92	0.82	0.77	0.86	0.79	0.65	0.75

[a] Johns Hopkins Turbulence Database [Perlman et al., 2007] [b] ScalarFlow [Eckert et al., 2019] [c] Combined test data sets

Table 1: Top: performance comparison of different metrics via the SRCC (**bold+underlined**: best method for each data set, **bold**: within a 0.01 margin of the best performing). Bottom: ablation study of the proposed method (gray: advantage due to different training data).

The bottom of Tab. 1 contains an ablation study of the proposed architecture *MS* and a simple *CNN* model that does not utilize a multiscale structure. Even though *VolSiM* has more than 80% fewer weights compared to $CNN_{trained}$, it can fit the training data more easily and generalizes better to most data sets, indicating the strengths of the proposed multiscale architecture. We replace the non-linear transformation of $\boldsymbol{w}_s$ from the similarity model with an identity transformation for $MS_{identity}$ during training, i.e. only the sequence order determines $\boldsymbol{g}$. This consistently lowers the generalization of the metric across data sets, indicating that well calibrated sequences as well as the similarity model are important. Removing the last resolution scale block for $MS_{3\ scales}$ overly reduces the capacity of the model, while adding another block for $MS_{5\ scales}$ is not beneficial. In addition, we also investigate two slightly different training setups: for $\mathrm{MS}_{\text{added}\ \mathtt{Iso}}$ we integrate extra sequences created like the `Iso` data during training, while $\mathrm{MS}_{\text{only}\ \mathtt{Iso}}$ is exclusively trained on such sequences. $\mathrm{MS}_{\text{added}\ \mathtt{Iso}}$ only slightly improves upon the baseline, and even the turbulence-specific $\mathrm{MS}_{\text{only}\ \mathtt{Iso}}$ model does not consistently improve the results on the JHTDB data sets. Both cases indicate a high level of generalization for *VolSiM*, as it was not trained on any turbulence data.

5 Conclusion

We presented the multiscale CNN architecture *VolSiM* that is trained with a similarity model based on the behavior of entropy in physical systems. Its capabilities as a metric for volumetric simulations were highlighted and utilized to learn a robust, physical similarity assessment. The proposed metric potentially has an impact on various disciplines where volumetric simulation data arises.

Ethical Statement

Since we target the fundamental problem of the similarity assessment of numerical simulations, we do not see any direct negative ethical implications of our work. However, there could be indirect negative effects since this work can act as a tool for more accurate and/or robust numerical simulations in the future, for which a military relevance exists. A further indirect issue could be explainability, e.g. when simulations in an engineering process yield unexpected inaccuracies.

References

L. Boltzmann. *Über Die Mechanische Bedeutung Des Zweiten Hauptsatzes Der Wärmetheorie: (Vorgelegt in Der Sitzung Am 8. Februar 1866)*. Staatsdruckerei, 1866.

M. Chu and N. Thuerey. 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.

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems 29*, volume 29, 2016. URL http://arxiv.org/abs/1602.02644.

M.-L. Eckert, K. Um, and N. Thuerey. 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.

P. Holl, N. Thuerey, and V. Koltun. Learning to control pdes with differentiable physics. In *8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HyeSin4FPB.

P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, 2012. ISBN 978-0-511-91970-1. doi:10.1017/CBO9780511919701.

A. Horé and D. Ziou. Image quality metrics: Psnr vs. ssim. In *20th International Conference on Pattern Recognition (ICPR 2010)*, pages 2366–2369, 2010. doi:10.1109/ICPR.2010.579.

Q. Huynh-Thu and M. Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics Letters*, 44(13):800–801, 2008. ISSN 0013-5194. doi:10.1049/el:20080522.

Q. Huynh-Thu and M. Ghanbari. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 49(1):35–48, 2012. ISSN 1572-9451. doi:10.1007/s11235-010-9351-x.

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision - ECCV 2016*, volume 9906, pages 694–711, 2016. doi:10.1007/978-3-319-46475-6_43.

G. Kohl, K. Um, and N. Thuerey. Learning similarity metrics for numerical simulations. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pages 5349–5360, 2020. URL http://proceedings.mlr.press/v119/kohl20a.html.

M. Meilă. Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007. ISSN 0047-259X. doi:10.1016/j.jmva.2006.11.013.

M. S. Olufsen, C. S. Peskin, W. Y. Kim, E. M. Pedersen, A. Nadim, and J. Larsen. Numerical simulation and experimental validation of blood flow in arteries with structured-tree outflow conditions. *Annals of Biomedical Engineering*, 28(11):1281–1299, 2000. doi:10.1114/1.1326031.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019. doi:10.48550/arXiv.1912.01703.

K. Pearson. Notes on the history of correlation. *Biometrika*, 13(1):25–45, 1920. doi:10.1093/biomet/13.1.25.

E. Perlman, R. Burns, Y. Li, and C. Meneveau. Data exploration of turbulence simulations using a database cluster. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing*, pages 1–11, 2007. doi:10.1145/1362622.1362654.

S. Pope. *Turbulent Flows*. Cambridge University Press, 2000. ISBN 978-0-511-84053-1. doi:10.1017/CBO9780511840531.

E. Prashnani, H. Cai, Y. Mostofi, and P. Sen. Pieapp: Perceptual image-error assessment through pairwise preference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1808–1817. IEEE Computer Society, 2018. doi:10.1109/CVPR.2018.00194.

A. Rényi. On measures of entropy and information. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561. University of California Press, 1961.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948. ISSN 0005-8580. doi:10.1002/j.1538-7305.1948.tb01338.x.

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. doi:10.2307/1412159.

T. Stocker, D. Qin, G.-K. Plattner, M. Tignor, S. Allen, J. Borschung, A. Nauels, Y. Xia, V. Bex, and P. Midgley. *Climate Change 2013: The Physical Science Basis*. Cambridge University Press, 2014. ISBN 978-1-107-41532-4. doi:10.1017/CBO9781107415324.

H. Talebi and P. Milanfar. Learned perceptual image enhancement. In *2018 IEEE International Conference on Computational Photography (ICCP)*, 2018. doi:10.1109/ICCPHOT.2018.8368474.

N. Thuerey and T. Pfaff. Mantaflow, 2018. URL `http://mantaflow.com`.

K. Um, X. Hu, and N. Thuerey. Perceptual evaluation of liquid simulation methods. *ACM Transactions on Graphics*, 36(4), 2017. doi:10.1145/3072959.3073633.

K. Um, X. Hu, B. Wang, and N. Thuerey. Spot the difference: Accuracy of numerical simulations via the human visual system. *ACM Transactions on Applied Perception*, 18(2):6:1–6:15, 2021. doi:10.1145/3449064.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. Simoncelli. 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.

R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. doi:10.1109/CVPR.2018.00068.

Y. Zhu and R. Bridson. Animating sand as a fluid. In *ACM Transactions on Graphics*, pages 965–972, 2005. doi:10.1145/1186822.1073298.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A, B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] All data sets contain a larger number of individual data pairs across sequences with different random initializations, leading to comparatively stable evaluations. In most cases an aggregation of multiple test sets is reported.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

APPENDIX

In the following, additional details for the proposed *VolSiM* metric are provided: App. A contains implementation details regarding the training and the metric model setup, and App. B features generation details and visualizations for all our data sets.

A Implementation Details

In addition to the multiscale feature extractor network, the following operations were used for the Siamese architecture of the metric: Each feature map is normalized via a mean and standard deviation normalization to a standard normal distribution. The mean and standard deviation of each feature map is computed in a pre-processing step for the initialization of the network over all data samples. Both values are fixed for training the metric afterwards. To compare both sets of feature maps in the latent space, a simple element-wise, squared difference is employed. To keep the mathematical metric properties, this also requires a square root operation before the final distance output. The spatial squared feature map differences are then aggregated along all dimensions into a scalar distance output. Here, we used a single learned weight with dropout for every feature map, to combine them to a weighted average per network layer. The activations of the average feature maps are spatially combined with a simple mean, and summed over all network layers afterwards. This process of normalizing, comparing, and aggregating the feature maps computed by the feature extractor follows previous work [Kohl et al., 2020, Zhang et al., 2018]. The weights to adjust the influence of each feature map are initialized to 0.1, all other weights of the multiscale feature extractor are initialized with the default PyTorch initialization. For the final loss, the MSE term was weighted with $\lambda_1 = 1.0$, while the correlation term was weighted with $\lambda_2 = 0.7$.

The training and evaluation process of the metric was implemented in PyTorch [Paszke et al., 2019], while the data was simulated and collected with specialized solvers and data interfaces as described in App B. The data acquisition, training, and metric evaluation was performed on a server with an Intel i7-6850 (3.60Ghz) CPU and an NVIDIA GeForce GTX 1080 Ti GPU. It took about 38 hours of training to fully optimize the final *VolSiM* model for the data sequences with a spatial resolution of 64^3. To increase the model's robustness during training, we used the following data augmentations for each sequence: the data is normalized to $[-1, 1]$, and together randomly flipped and rotated in increments of 90° around a random axis. The velocity channels are randomly swapped to prevent directional biases from some simulations, while scalar data is extended to the three input channels via repetition. For inference, only the normalization operation and the repetition of scalar data is performed. The final metric model was trained with the Adam optimizer with a learning rate of 10^{-4} for 30 epochs via early stopping.

B Data Set Details

In the following sections, the details underlying each data set are described. Tab. 2 contains a summary of simulator, simulation setup, varied parameters, noise integration, and used fields for the simulated and generated data sets. Tab. 3 features a summary of the collected data sets, with repository details, jitter and cutout settings, and spatial and temporal Δ values. Both tables also contain the number of sequences created for training, validation, and testing for every data source.

B.1 Advection-Diffusion and Burgers' Equation

In its simplest form, the transport of matter in a flow can be described by the two phenomena of advection and diffusion. Advection describes the movement of a passive quantity inside a velocity field over time, and diffusion describes the process of dissipation of this quantity due to the second law of thermodynamics.

$$\frac{\partial d}{\partial t} = \nu \nabla^2 d - u \cdot \nabla d \qquad (3)$$

Eq. 3 is the simplified Advection-Diffusion equation with constant diffusivity and no sources or sinks, where u denotes the velocity, d is a scalar passive quantity that is transported, and ν is the diffusion coefficient or kinematic viscosity.

Burgers' Equation in Eq. 4 is similar to the Advection-Diffusion equation, but it describes how the velocity field itself changes over time with advection and diffusion. The diffusion term can also be interpreted as a viscosity, that models the resistance of the material to deformations. Furthermore, this variation can develop discontinuities (also called shock waves). Here, u also denotes the velocity and ν the kinematic viscosity or diffusion coefficient.

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - u \cdot \nabla u \tag{4}$$

To solve both PDEs, the differentiable fluid framework PhiFlow [Holl et al., 2020] was used. The solver utilizes a Semi-Lagrangian advection scheme, and we chose periodic domain boundary conditions to allow for the usage of a Fourier space diffusion solver. We introduced additional continuous forcing to the simulations by adding a force term f to the velocity after every simulation step. Thus, f depends on the time steps t, that is normalized by division of the simulation domain size beforehand. For `Adv`, `Bur`, and `AdvD`, we initialized the fields for velocity, density, and force with multiple layered parameterized sine functions. This leads to a large range of patterns across multiple scales and frequencies when varying the sine parameters.

$$u^x(\boldsymbol{p}) = \mathrm{sum}\left(\boldsymbol{f}_1^x + \sum_{i=1}^{4} \boldsymbol{f}_{i+1}^x * \sin(2^i \pi \boldsymbol{p} + c_i\, \boldsymbol{o}^x_{(i+1 \bmod 2)+1})\right) \qquad \text{where } \boldsymbol{c} = (1,\, 1,\, 0.4,\, 0.3) \tag{5}$$

$$f^x(\boldsymbol{p}, t) = \mathrm{sum}\left(\boldsymbol{f}_6^x * (1 + \boldsymbol{f}_6^x * 20) * \sum_{i=1}^{4} \boldsymbol{f}_{i+1}^x * \sin(2^i \pi \tilde{\boldsymbol{p}} + c_i\, \boldsymbol{o}^x_{(i \bmod 2)+1})\right) \qquad \text{where } \tilde{\boldsymbol{p}} = \boldsymbol{p} + \boldsymbol{f}_7^x * 0.5 + \boldsymbol{f}_7^x * \sin(3t) \text{ and } \boldsymbol{c} = (0,\, 1,\, 1,\, 0.7) \tag{6}$$

$$d(\boldsymbol{p}) = \mathrm{sum}\left(\sum_{i}^{\{x,y,z\}} \sin(\boldsymbol{f}_d^i * 24\pi p_i + o_d^i)\right) \tag{7}$$

Eq. 5, 6, and 7 show the layered sine functions in $u^x(\boldsymbol{p})$, $f^x(\boldsymbol{p})$, and $d(\boldsymbol{p})$ for a spatial grid position $\boldsymbol{p} \in \mathbb{R}^3$. The sum operation denotes a sum of all vector elements here, and all binary operations on vectors and scalars use broadcasting of the scalar value to match the dimensions. Eq. 8 shows the definition of the function parameters used above, all of which are randomly sampled based on the simulation seed for more diverse simulations.

$$\begin{array}{llll} \boldsymbol{f}_1^x \sim \mathcal{U}(-0.2, 0.2)^3 & \boldsymbol{f}_4^x \sim \mathcal{U}(-0.15, 0.15)^3 & \boldsymbol{f}_7^x \sim \mathcal{U}(-0.1, 0.1)^3 & \boldsymbol{o}_1^x \sim \mathcal{U}(0, 100)^3 \\ \boldsymbol{f}_2^x \sim \mathcal{U}(-0.2, 0.2)^3 & \boldsymbol{f}_5^x \sim \mathcal{U}(-0.1, 0.1)^3 & \boldsymbol{f}_d^{x,y,z} \sim \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}\}^3 & \boldsymbol{o}_2^x \sim \mathcal{U}(0, 100)^3 \\ \boldsymbol{f}_3^x \sim \mathcal{U}(-0.15, 0.15)^3 & \boldsymbol{f}_6^x \sim \mathcal{U}(0.0, 0.1)^3 & \nu \sim \mathcal{U}(0.0002, 0.1002) & \boldsymbol{o}_d^{x,y,z} \sim \mathcal{U}(0, 100)^3 \end{array} \tag{8}$$

Note that ν was multiplied by 0.1 for `Bur`. The remaining velocity and force components $u^y(\boldsymbol{p})$, $u^z(\boldsymbol{p})$, $f^y(\boldsymbol{p})$, and $f^z(\boldsymbol{p})$ and corresponding parameters are omitted for brevity here, since they follow the same initialization pattern as $u^x(\boldsymbol{p})$ and $f^x(\boldsymbol{p})$. Tab. 2 shows the function parameters that were varied, by using the random initializations and adjusting one of them in linear steps to create a sequence. The main difference between the Advection-Diffusion training data and the test set is the method of noise integration: For `Adv` it is integrated into the simulation velocity, while for `AdvD` it is added to the density field instead. The amount of noise added to the velocity for `Bur` and `Adv` and to the density for `AdvD` was varied in isolation as well.

B.2 Navier-Stokes Equations

The Navier-Stokes Equations fully describe the behavior of fluids like gases and liquids, via modelling advection, viscosity, and pressure effects, as well as mass conservation. Pressure can be interpreted as the force exerted by surrounding fluid mass at a given point, and the conservation of mass means that the fluid resists compression.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla) u = -\frac{\nabla P}{\rho} + \nu \nabla^2 u + g \tag{9}$$

$$\nabla \cdot u = 0. \tag{10}$$

Eq. 9 describes the conservation of momentum, and Eq. 10 describes mass conservation. Again, u denotes the velocity, P is the pressure, ρ is the fluids density, ν is the kinematic viscosity, and g denotes external forces like gravity.

Smoke To create the smoke data set `Smo`, the fluid framework MantaFlow [Thuerey and Pfaff, 2018] that provides a grid-based Eulerian smoke solver for the Navier-Stokes Equations was used. It is based on a Semi-Lagrangian advection scheme, and on the conjugate gradient method as a pressure solver. The simulation setup consists of a cylindrical smoke source at the bottom of the domain with a fixed noise pattern initialization to create more diverse smoke plumes. Furthermore, a constant spherical force field *ff* is positioned over the source. This setup allows for a variation of multiple simulation parameters, like the smoke buoyancy, the source position and different force field settings. They include position, rotation, radius and strength. In addition, the amount of added noise to the velocity can also be varied in isolation.

Liquid Both liquid data sets, `Liq` and `LiqN`, were created with a liquid solver in MantaFlow. It utilizes the hybrid Eulerian-Lagrangian fluid implicit particle method [Zhu and Bridson, 2005], that combines the advantages of particle and grid-based liquid simulations for reduced numerical dissipation. The simulation setup consists of two liquid cuboids of different shapes, similar to the common breaking dam setup. After 25 simulation time steps a liquid drop is added near the top of the simulation domain, and it falls down on the water surface that is still moving. Here, the external gravity force as well as the drops position and radius are varied to create similarity sequences. As for the smoke data, a modification of the amount of noise added to the velocity was also employed as a varied parameter. The main difference between the liquid training data and the test set is the method of noise integration: For `Liq` it is integrated into the simulation velocity, while for `LiqN` it is overlayed on the simulation background.

B.3 Generated Data

To create the shape data set `Sha` and the wave data set `Wav`, a random number of straight paths are created by randomly generating a start and end point inside the domain. It is ensured that both are not too close to the boundaries and that the path has a sufficient length. The intermediary positions for the sequence are a result of linearly interpolating on these paths. The positions on the path determine the center for the generated objects that are added to an occupancy marker grid. For both data sets, overlapping shapes and waves are combined additively, and variations with and without overlayed noise to the marker grid were created.

Shapes For `Sha`, random shapes (box or sphere) are added to the positions, where the shape's size is a random fraction of the path length, with a minimum and maximum constraint. The created shapes are then applied to the marker grid either with or without smoothed borders.

Waves For `Wav`, randomized volumetric damped cosine waves are added around the positions instead. The marker grid value m at point $\boldsymbol{p}$ for a single wave around a center $\boldsymbol{c}$ is defined as

$$m(\boldsymbol{p}) = cos(w * \tilde{p}) * e^{-(3.7\tilde{p}/r)} \text{ where } \tilde{p} = \|\boldsymbol{p} - \boldsymbol{c}\|_2 \,.$$

Here, r is the radius given by the randomized size that is computed as for `Sha`, and $w \sim \mathcal{U}(0.1, 0.3)$ is a randomized waviness value, that determines the frequency of the damped wave.

B.4 Collected Data

The collected data sets `Iso`, `Cha`, `Mhd`, and `Tra` are based on different subsets from the Johns Hopkins Turbulence Database JHTDB [Perlman et al., 2007], that contain different types of data from direct numerical simulations (DNS): isotropic turbulence (`Iso`), a channel flow (`Cha`), magneto-hydrodynamic turbulence (`Mhd`), and a transitional boundary layer (`Tra`). In these simulations, all spatial scales of turbulence up to the dissipative regime are resolved. The data set `SF` is based on the ScalarFlow data [Eckert et al., 2019], that contains dense 3D velocity fields of real buoyant smoke plumes, created via multi-view reconstruction technique.

JHTDB The JHTDB subsets typically contain a single simulation with a very high spatial and temporal resolution and a variety of fields. We focus on the velocity fields, since turbulent flow data is especially complex and potentially benefits most from a better similarity assessment. We can mainly rely on using temporal sequences, and only need to add spatial jitters in some cases to increase the difficulty. As turbulence generally features structures of interest across all length scales, we create sequences of different spatial scales for each subset. To achieve this, we randomly pick a cutout scale factor s. If $s = 1$, we directly use the native spatial discretization provided by the database. For $s > 1$ we stride the spatial query points of the normal cubical cutout by s after filtering the data. For $s < 1$ the size of the cubical cutout is reduced by a factor of s in each dimension, and the cutout is interpolated to the full spatial size of 128^3 afterwards. Among other details, Tab. 3 shows the cutout scale factors, as well as the corresponding random weights.

ScalarFlow Since 100 reconstructions of different smoke plumes are provided in ScalarFlow, there is no need to add additional randomization to create multiple test sequences. Instead, we directly use each reconstruction sequence to create one similarity sequence in equal temporal steps. The only necessary pre-processing step is cutting off the bottom part of domain that contains the smoke inflow, since it is frequently not fully reconstructed. Afterwards, the data is interpolated to the full spatial size of 128^3 to match the other data sets.

B.5 Additional Example Sequences

Fig. 4, 5, and 6 show multiple full example sequences from all our data sets. In every sequence, the leftmost image is the baseline field. Moving further to the right, the change of one initial parameter increases for simulated data sets, and the spatio-temporal position offset increases for generated and collected data. To plot the sequences, the 3D data is projected along the z-axis to 2D via a simple mean operation. This means, noise that was added to the data or the simulation is typically significantly less obvious due to statistical averaging in the projection. Velocity data is directly mapped to RGB color channels, and scalar data is shown via different shades of gray. Unless note otherwise, the data is jointly normalized to $[0, 1]$ for all channels at the same time, via the overall minimum and maximum of the data field.

	`Adv`	`Bur`	`Liq`	`Smo`	`AdvD`	`LiqN`	`Sha`	`Wav`
Sequences train–val–test	398–57–0	408–51–0	405–45–0	432–48–0	0–0–57	0–0–30	0–0–60	0–0–60
Equation	Eq. 3	Eq. 4	Eq. 9, 10	Eq. 9, 10	Eq. 3	Eq. 9, 10	—	—
Simulator	PhiFlow[d]	PhiFlow[d]	MantaFlow[e]	MantaFlow[e]	PhiFlow[d]	MantaFlow[e]	MantaFlow[e]	MantaFlow[e]
Simulation setup	layered sines	layered sines	breaking dam + drop	rising plume with force field	layered sines	breaking dam + drop	random shapes	random damped waves
Time steps	120	120	80	120	120	80	—	—
Varied aspects	$f_1, f_2,$ $f_3, f_4,$ $f_5, f_7,$ $o_1, o_2,$ $o_d, noise$	$f_1, f_2,$ $f_3, f_4,$ $f_5, f_7,$ $o_1, o_2,$ $noise$	$drop_x$ $drop_y$ $drop_z$ $drop_{rad}$ $grav_x$ $grav_y$ $grav_z$ $noise$	$buoy_x$ $buoy_y$ $ff_{rot\ x}$ $ff_{rot\ z}$ $ff_{str\ x}$ $ff_{str\ z}$ $ff_{pos\ x}$ $ff_{pos\ y}$ ff_{rad} $source_x$ $source_y$ $noise$	$f_1, f_2,$ $f_3, f_4,$ $f_5, f_7,$ $o_1, o_2,$ $o_d, noise$	$drop_x$ $drop_y$ $drop_z$ $drop_{rad}$ $grav_x$ $grav_y$ $grav_z$ $noise$	shape position	wave position
Noise integration	added to velocity	added to velocity	added to velocity	added to velocity	added to density	overlay on non-liquid	overlay on marker	overlay on marker
Used fields	density	velocity	velocity flags levelset	density pressure velocity	density	velocity	marker	marker

[d] PhiFlow from Holl et al. [2020] [e] MantaFlow from Thuerey and Pfaff [2018]

Table 2: Data set detail summary for the simulated and generated data sets.

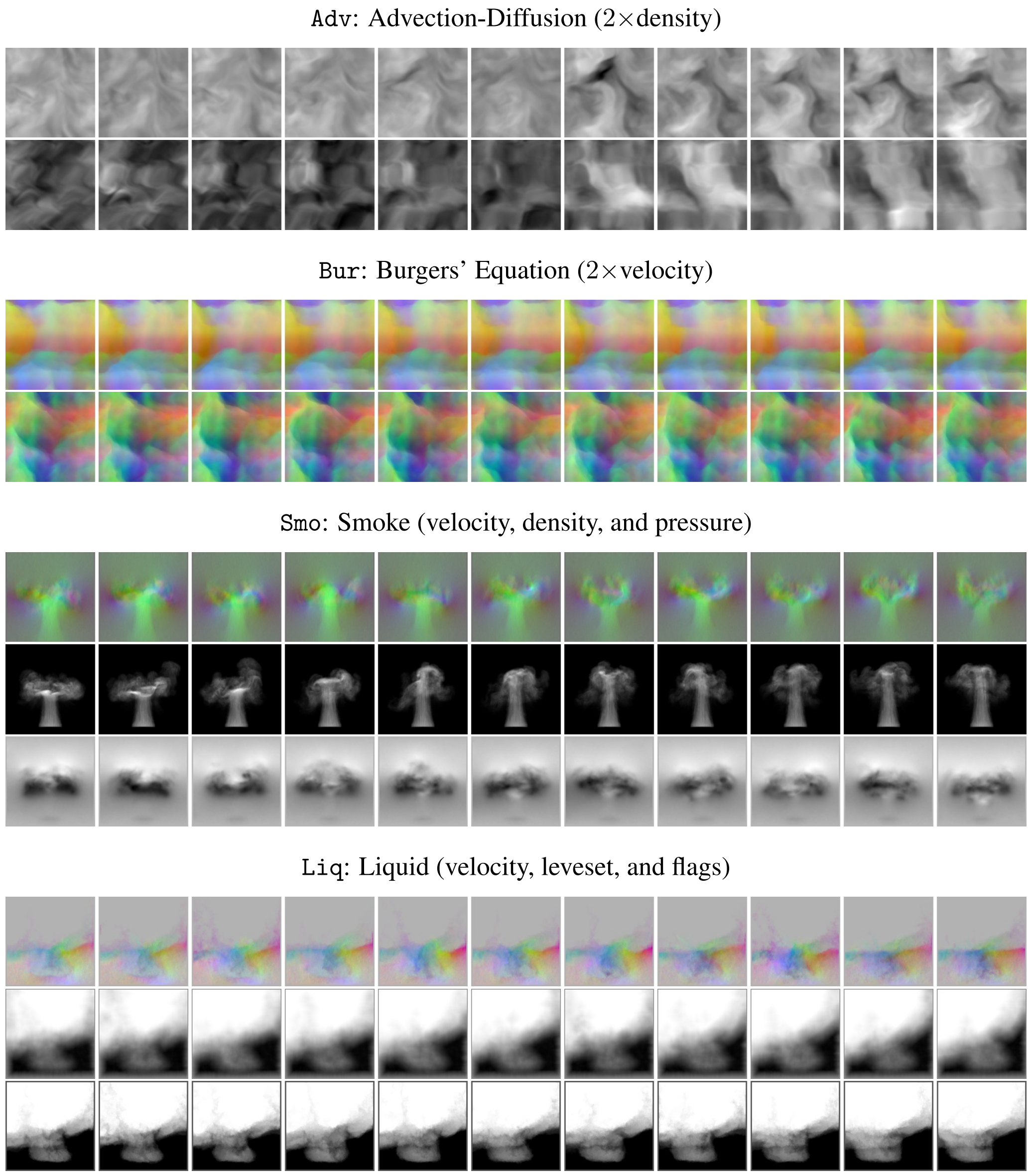

Adv: Advection-Diffusion (2×density)
Bur: Burgers' Equation (2×velocity)
Smo: Smoke (velocity, density, and pressure)
Liq: Liquid (velocity, leveset, and flags)

Figure 4: Example sequences of simulated training data, where each row features a full sequence from a different random seed.

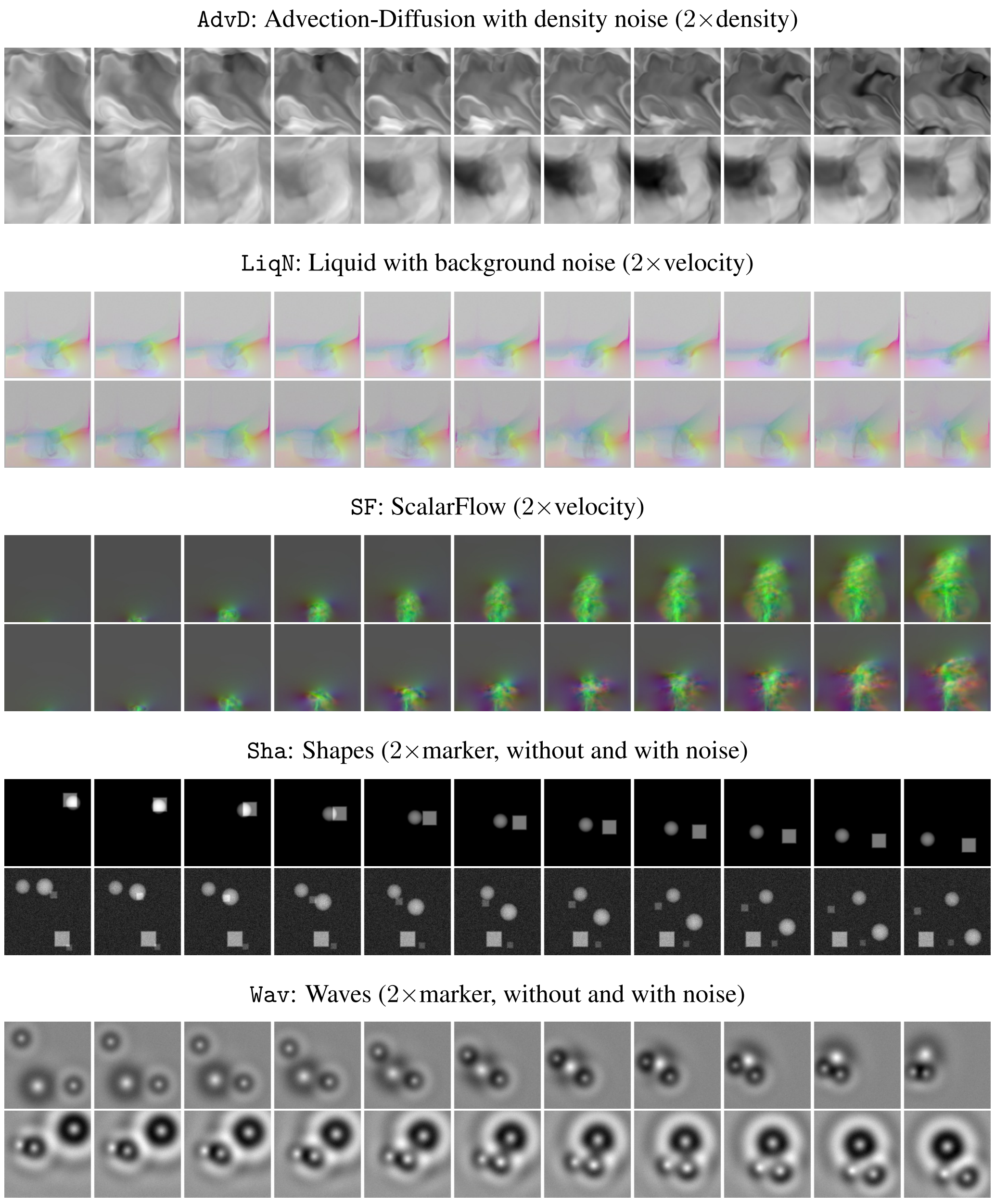

AdvD: Advection-Diffusion with density noise (2×density)
LiqN: Liquid with background noise (2×velocity)
SF: ScalarFlow (2×velocity)
Sha: Shapes (2×marker, without and with noise)
Wav: Waves (2×marker, without and with noise)

Figure 5: Example sequences of simulated (top two data sets), collected (middle data set), and generated (bottom two data sets) test data. Each row contains a full sequence from a different random seed. It is difficult to visually observe the background noise in `LiqN` due the projection along the z-axis to 2D, and due to image compression.

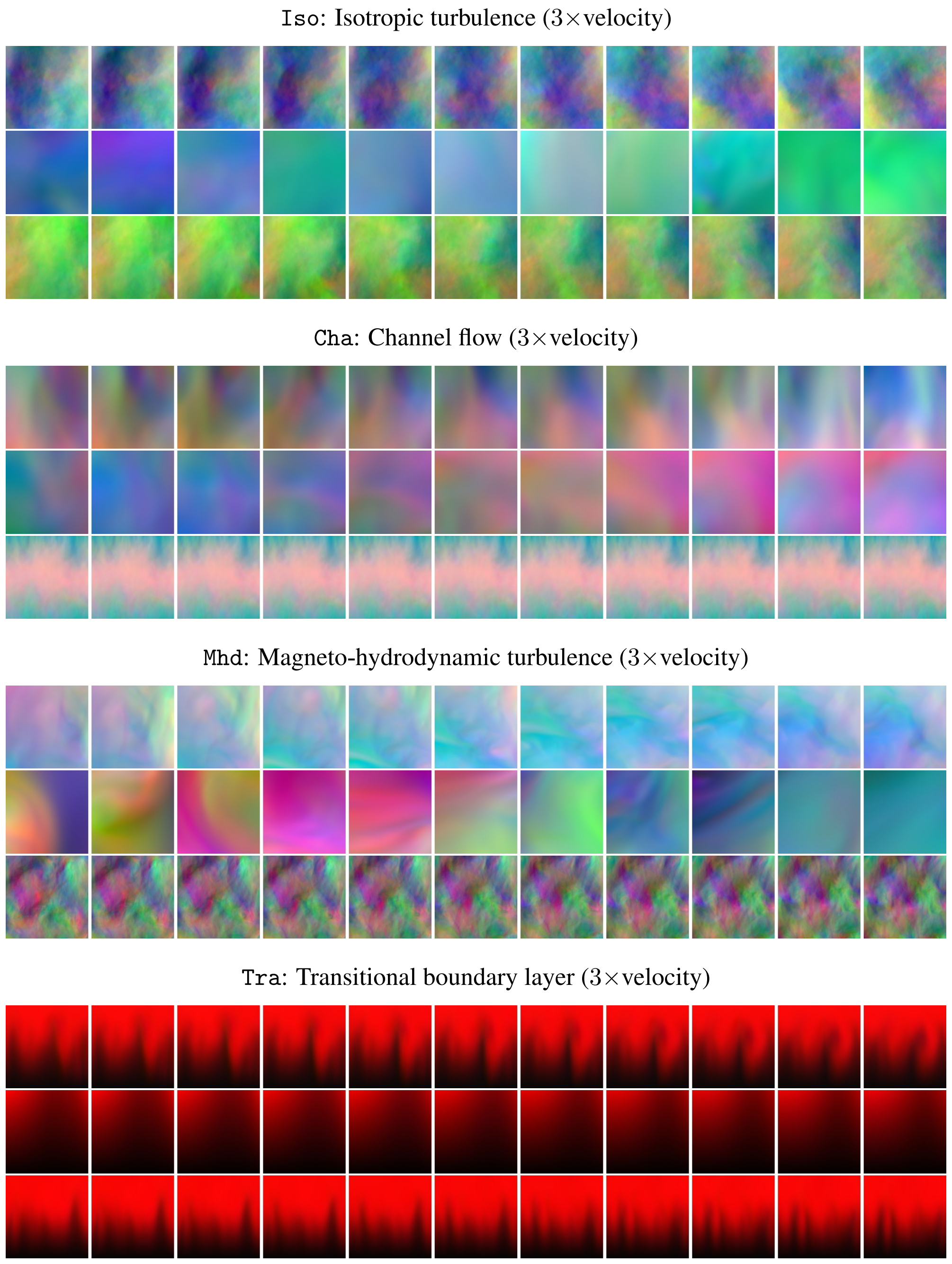

Iso: Isotropic turbulence (3×velocity)
Cha: Channel flow (3×velocity)
Mhd: Magneto-hydrodynamic turbulence (3×velocity)
Tra: Transitional boundary layer (3×velocity)

Figure 6: Example sequences of collected test data from JHTDB, where each row shows a full sequence from a different random seed. Notice the smaller cutout scale factor s for the middle example in each case. The predominant x-component in `Cha` is separately normalized for a more clear visualization.

	`Iso`	`Cha`	`Mhd`	`Tra`	`SF`
Sequences train–val–test	0–0–60	0–0–60	0–0–60	0–0–60	0–0–100
Repository	JHTDB – isotropic 1024coarse [f]	JHTDB – channel [f]	JHTDB – mhd1024 [f]	JHTDB – transition_bl [f]	ScalarFlow [g]
Repository size [h] $s \times t \times x \times y \times z$	$1 \times 5028 \times 1024 \times 1024 \times 1024$	$1 \times 4000 \times 2048 \times 512 \times 1536$	$1 \times 1024 \times 1024 \times 1024 \times 1024$	$1 \times 4701 \times 10240 \times 1536 \times 2048$	$100 \times 150 \times 100 \times 178 \times 100$ [i]
Temporal offset Δ_t	180	37	95	25	13
Spatial offset $\Delta_{x,y,z}$	0	0	0	0	0
Spatial jitter	0	0	25	0	0
Cutout scales	$0.25, 0.5, 0.75, 1, 2, 3, 4$	$0.25, 0.5, 0.75, 1, 2, 3, 4$	$0.25, 0.5, 0.75, 1, 2, 3, 4$	$0.25, 0.5, 0.75, 1, 2$	1
Cutout scale random weights	$0.14, 0.14, 0.14, 0.16, 0.14, 0.14, 0.14$	$0.14, 0.14, 0.14, 0.16, 0.14, 0.14, 0.14$	$0.14, 0.14, 0.14, 0.16, 0.14, 0.14, 0.14$	$0.14, 0.14, 0.14, 0.30, 0.28$	1
Used fields	velocity	velocity	velocity	velocity	velocity

[f] JHTDB from Perlman et al. [2007] [g] ScalarFlow from Eckert et al. [2019] [h] simulations s × time steps t × spatial dimensions x, y, z [i] cut to $100 \times 150 \times 100 \times 160 \times 100$ (removing 18 bottom values from y), since the smoke inflow is not fully reconstructed

Table 3: Data set detail summary for collected data sets.