ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020
J. Bender and T. Popa
(Guest Editors)

Volume 39 (2020), Number 8

Latent Space Subdivision: Stable and Controllable Time Predictions for Fluid Flow

S. Wiewel[1], B. Kim[2], V. C. Azevedo[2], B. Solenthaler[2], N. Thuerey[1] †

[1]Technical University of Munich
[2]ETH Zurich

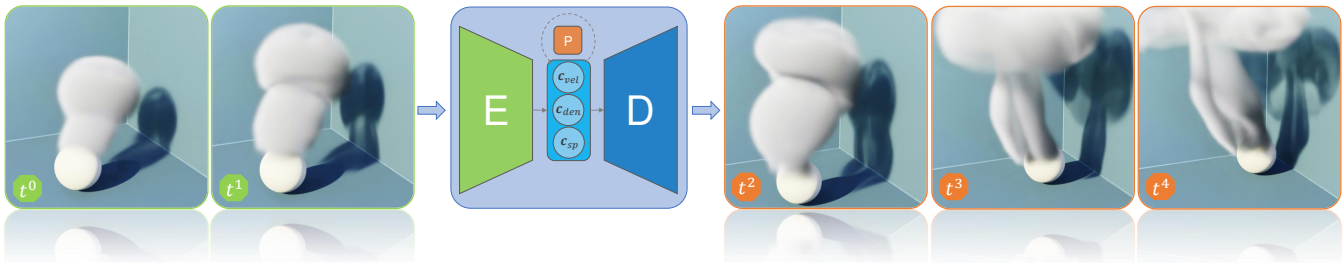


Figure 1: *Given two consecutive data points (t_0, t_1) of a fluid simulation, a stable long-term prediction of complex 3D flows is generated (t_2, t_3, t_4) from the compressed information in the latent space. Thereby, our novel split loss allows for modification and control of the learned-physics predictor, e.g., by injecting a density state resulting from an external advection step into the latent space part $\mathbf{c}_{den}$. This density field contains, for example, temporally coherent data and proper boundary conditions for moving obstacles or inflows.*

Abstract

We propose an end-to-end trained neural network architecture to robustly predict the complex dynamics of fluid flows with high temporal stability. We focus on single-phase smoke simulations in 2D and 3D based on the incompressible Navier-Stokes (NS) equations, which are relevant for a wide range of practical problems. To achieve stable predictions for long-term flow sequences with linear execution times, a convolutional neural network (CNN) is trained for spatial compression in combination with a temporal prediction network that consists of stacked Long Short-Term Memory (LSTM) layers. Our core contribution is a novel latent space subdivision (LSS) to separate the respective input quantities into individual parts of the encoded latent space domain. As a result, this allows to distinctively alter the encoded quantities without interfering with the remaining latent space values and hence maximizes external control. By selectively overwriting parts of the predicted latent space points, our proposed method is capable to robustly predict long-term sequences of complex physics problems, like the flow of fluids. In addition, we highlight the benefits of a recurrent training on the latent space creation, which is performed by the spatial compression network. Furthermore, we thoroughly evaluate and discuss several different components of our method.

CCS Concepts

• ***Computing methodologies*** → ***Neural networks; Physical simulation;***

1. Introduction

Computing the dynamics of fluids requires solving a set of complex equations over time. This process is computationally very expensive, especially when considering that the stability requirement poses a constraint on the maximal time step size that can be used in a simulation.

Due to the high computational resources, approaches for machine learning based physics simulations have recently been explored. One of the first approaches used Regression Forest as a regressor to forward the state of a fluid over time [LJS*15]. Handcrafted features have been used, representing the individual terms of the Navier-Stokes equations. These context-based integral features can be evaluated in constant time and robustly forward the

† This work was supported by the ERC Starting Grant *realFlow* (StG-2015637014) and the Swiss National Science Foundation (grant no. 200021_168997). Source code and video: https://ge.in.tum.de/publications/2020-lssubdiv-wiewel/

state of the system over time. In contrast, using neural networks for the time prediction has the advantage that no features have to be defined manually, and hence these methods have recently gained increased attention. In graphics, the presented neural prediction methods [MJKW18; KCT*19; WBT19] use a two-step approach, where first the physics fields are translated into a compressed representation, i.e., the latent space. Then, a second network is used to predict the state of the system over time in the latent space. The two networks are trained individually, which is an intuitive approach as spatial and temporal representations can be separated by design. In practice, the first network (i.e., the autoencoder) introduces small errors in the encoding and decoding in each time step. In combination with a temporal prediction network these errors accumulate over time, introducing drifting over prolonged time spans and can even lead to instability, as we will show later. This is especially problematic in supervised learned latent space representations, since the drift will shift the initial, user-specified conditions (e.g., an object's position) into an erroneous latent space configuration.

Like previous work, we use a neural network to predict the motion of a fluid over time, but with the central goal to increase accuracy and robustness of long-term predictions. We propose to use a *joint end-to-end training* of both components: the fluid state compression and the temporal prediction of the motion. Such a joint training allows both components to build a holistic view of the underlying task, which is not the case for individual training. The joint training enables the network to propagate the error gradients from the spatial decoder through multiple recurrently connected temporal prediction blocks to the spatial encoder. Another key observation is the need to *control the learned latent space*, such that we can modify it during the simulation process to impose boundary conditions and other known information external to the simulation state. Without such control capabilities, the underlying prediction model cannot react to external changes in the simulation domain. We therefore propose structuring the latent space through subdivisions, which separate the encoded quantities in the latent space domain. This enables us to directly manipulate the individual quantities in the encoded latent space domain and therefore, e.g., feed back a density field advected externally with the predicted velocity field. The subdivision is enforced with a *split latent space soft-constraint* for the input quantities velocity and density, and hence allows for alteration of the individual encoded components separately. This separation is a key component to robustly predict long-term sequences as demonstrated by our results. It enables us to precisely adjust parts of the latent space to adhere to physical behavior determined externally by well-understood classical methods.

2. Related Work and Background

Our work concentrates on single-phase flows, which are usually modeled by a pressure-velocity formulation of the incompressible Navier-Stokes equations:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = \frac{-1}{\rho_0}\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{g} \qquad (1)$$
$$\nabla\cdot\boldsymbol{u} = 0,$$

where p denotes pressure, $\boldsymbol{u}$ the flow velocity, and $\rho_0, \nu, \boldsymbol{g}$ denote the density of the fluid (which is assumed to be constant), kinematic viscosity, and external forces, respectively. Often, a passive marker quantity ρ is advected with the velocity $\boldsymbol{u}$, e.g., to model smoke. An overview of fluid simulation techniques in computer graphics can be found in [Bri15].

Data-driven flow modelling encompasses two distinguished and complementary efforts: dimensionality reduction and reduced-order modelling. Dimensionality reduction (e.g., Singular Value Decomposition) scales down the analyzed data into a set of important features in an attempt to increase the sparsity of the representation, while reduced-order modelling (e.g., Galerkin Projection) describes the spatial and temporal dynamics of a system represented by a set of reduced parameters. In computer graphics, the work of Treuille et al. [TLP06] was the first to use Principal Component Analysis (PCA) for dimensionality reduction coupled with a Galerkin Projection method for subspace simulation. This approach was later extended [KD13] with a cubature approach for enabling semi-Lagrangian and MacCormack [SFK*08] advection schemes, while improving the handling of boundary conditions. Reduced-order modelling was also explored to accelerate the pressure projection step in liquid simulations [ATW15].

Instead of computing reduced representations from pre-simulated velocity fields, alternative basis functions can be used for reduced-order modelling; examples of basis functions include Legendre Polynomials [GN07], modular [WST09] and spectral [LR09] representations. Also Laplacian Eigenfunctions have been successfully employed for dimensionality reduction, due to their natural sparsity and inherent incompressibility. De Witt et al. [DLF12] combined Laplacian Eigenfunctions with a Galerkin Projection method, enabling fast and energy-preserving fluid simulations. The approach was extended to handle arbitrarily-shaped domains [LMH*15], combined with a Discrete Cosine Transform (DCT) for compression [JSK16], and improved for scalability [CSK18].

A separate line of work proposed direction interpolations of flow representations in order to synthesize new instances [RWTT14; Thu16; SDN18]. For particle-based fluid simulations, a temporal state prediction using Regression Forest was presented in Ladicky et al. [LJS*15]. Input features are evaluated in particle neighborhoods and serve as input to the regressor, which then predicts the particle velocity of the next time step.

Several of the methods above use linear basis functions for dimensionality reduction. This enables the use of Galerkin Projection for subspace integration, but it limits the power of the reconstruction when compared to non-linear embeddings. The latent spaces generated by autoencoders (AE) are non-linear and richly capture the input space with fewer variables [RMC16; WZX*16]. In light of that, Wiewel et al. [WBT19] combined a latent space representation with recurrent neural networks (RNN) to predict the temporal evolution of fluid functions in the latent space domain. Kim et al. [KCT*19] introduced a generative deep neural network for parameterized fluid simulations that only takes a small set of physical parameters as input to very efficiently synthesize points in the learned parameter space. Their method also proposes an extension to latent space integration by training a fully connected neural net-

work that maps subsequent latent spaces. Our work is related to these two methods, but a main difference is that we use an end-to-end training of both the spatial compression and the temporal prediction. In combination with our latent space subdivision, our predictions are more stable, while previous approaches fail to properly recover long-term integration correspondences due to the lack of autoencoder regularization.

In the context of grid-based (Eulerian) fluid simulations, Tompson et al. [TSSP17] used a convolutional neural network (CNN) to model spatial dependencies in conjunction with an unsupervised loss function formulation to infer pressure fields. A simpler three-layer fully connected neural network for the same goal was likewise proposed [YYX16]. As an alternative, learned time evolutions for Koopman operators were proposed [MJKW18], which however employ a pre-computed dimensionality reduction via PCA. Chu et al. [CT17] enhance coarse fluid simulations to generate highly detailed turbulent flows. Individual low-resolution fluid patches were tracked and mapped to high-resolution counterparts via learned descriptors. Xie et al. [XFCT18] extended this approach by using a conditional generative adversarial network with a spatio-temporal discriminator supervision. Small-scale splash details in hybrid fluid solvers were targeted with deep learning-based stochastic models [UHT18].

3. Method

The central goal of our models is to robustly and accurately predict long-term sequences of flow dynamics. For this, we need an autoencoder to translate high-dimensional physics fields into a compressed representation (latent space) and a temporal prediction network to advance the state of the simulation over time. A key observation is that if these two network components are trained individually, neither component has a holistic view on the underlying problem. The autoencoder, consisting of an encoder E and a decoder D, generates a compressed representation $\boldsymbol{c} = E(\boldsymbol{x})$, which focuses solely on the reconstruction $\tilde{\boldsymbol{x}} = D(\boldsymbol{c})$ of the given input $\boldsymbol{x}$. Hence, the loss function to minimize is given by $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$. Without considering the aspect of time, the autoencoder's latent space only stores spatial descriptors. Due to the exclusive focus on space, temporally consecutive data points are not necessarily placed close to each other in the latent space domain. This poses substantial challenges for the temporal prediction network.

Therefore, we consider the aspect of time within the training of the autoencoder in order to shape its latent space with respect to temporal information, in addition to the spatial information. Thus, we propose an end-to-end training procedure, where we train our autoencoder and temporal prediction network simultaneously by internally connecting the latter as a recurrent block to the encoding and decoding blocks of the spatial autoencoder. As a result, the latent space domain is aware of temporal changes, and can yield temporally coherent latent space points that are suitable for the time prediction network. By default, our training process includes the combined training of our spatial autoencoder and temporal prediction network as shown in Figure 2 and Figure 3, respectively. In those figures, the encoder E, decoder D and prediction network P are duplicated for visualization purposes. In the next sections, we describe each individual network in more detail.

3.1. Spatial Encoding

The spatial encoding of the data is performed by a regular autoencoder, the network for which is split into an encoding and decoding part [KCT*19]. The encoder contains 16 convolution layers with skip connections, which connect its internal layers, followed by one fully-connected layer. The decoder consists of a fully-connected layer, which is followed by 17 convolution layers with skip connections. For the fluid simulation dataset used in this work, the input is either 2D or 3D, leading to the usage of 2D- or 3D-convolutions and a feature dimension of 3 or 4, respectively. Furthermore, a data-specific curl-layer is appended to the decoder network to enforce zero divergence in the resulting velocity field [KCT*19], as required by the NS equations (see Equation 1).

The dimensionality of the latent space $\boldsymbol{c}$ for a given $\boldsymbol{x}$ is defined by the final layer of the encoder and can be freely chosen. We pass a velocity $\boldsymbol{u}$ as well as the density of a passive marker field ρ to our encoder, i.e., $\boldsymbol{x} = [\boldsymbol{u}, \rho]$. Note that this marker density is different from the (constant) density of the fluid itself. The velocity field is an active quantity that is used in fluid simulations to advect a passive quantity forward in time. In case of smoke simulations, which is a specific instance of fluid simulations, the passive density field is advected by the flow velocity. As a result, $\boldsymbol{c}$ contains information about both the active and passive fields, i.e., velocity and density. Hence, we can accurately advect the passive quantity density with a velocity field with low computational effort, it makes sense to

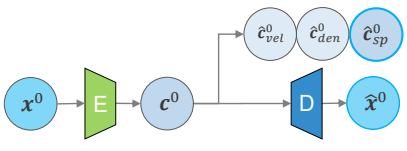


Figure 2: *A direct AE reconstruction loss, Equation 5, is performed on $\hat{\boldsymbol{x}}^0$, whereas the supervised parameter loss, Equation 4, is performed on $\hat{\boldsymbol{c}}^0_{sp}$. The superscript denotes the time step of the input data used during training.*

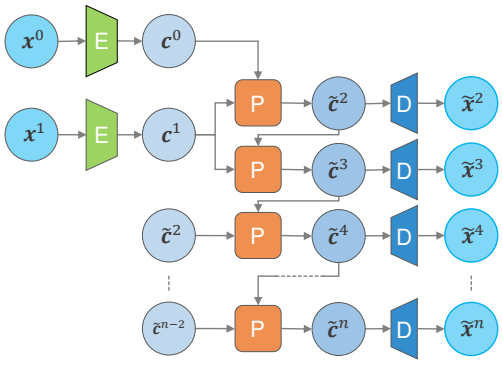


Figure 3: *A prediction loss, last term of Equation 6, is performed on $\tilde{\boldsymbol{x}}^2 \ldots \tilde{\boldsymbol{x}}^n$. The prediction networks input window is set to $w = 2$. Thus, the count of recurrent predictions is $n_i = n - 2$.*

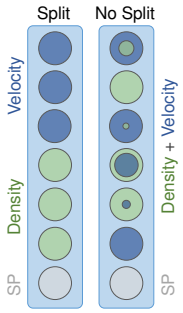
compute the advection outside of the network and project the new state into the latent space.

In order to be able to alter active and passive fields individually in the compressed representation $\boldsymbol{c}$, with given input field $\boldsymbol{x} = [\boldsymbol{x}_{vel}, \boldsymbol{x}_{den}]$ where the subscripts *vel* and *den* thereby denote the velocity and density part, we subdivide $\boldsymbol{c}$ into separate parts for velocity $\boldsymbol{c}_{vel}$ and density $\boldsymbol{c}_{den}$, respectively. This property of the latent space is needed for projecting the new state of the passive quantity into the latent space domain. Additionally, to exert explicit external control over the prediction, we designate another part of $\boldsymbol{c}$ to contain supervised parameters, called $\boldsymbol{c}_{sp}$ as proposed by Kim et al. [KCT*19]. In our case of, e.g., a smoke simulation with a rotating cup filled with smoke, such supervised parameters can be the position of a smoke source or the rotation angle of a solid obstacle. With this subdivision, we increase the stability of our predictions and allow for explicit external control. This subdivision is visualized in Figure 4 and Equation 2, where v, d, and sp describe the indices of the velocity, density, and supervised parameter parts in the latent space domain, respectively.

To arrive at our desired latent space subdivision (LSS), the split loss $\mathcal{L}_{split}$ is used as a loss function in the training process. It is modelled as a soft constraint and thereby does not enforce the parts $\boldsymbol{c}_{vel}$ and $\boldsymbol{c}_{den}$ to be strictly disjunct, hence separated by the $\wr$ symbol in Equation 2. The loss is defined as

$$\mathcal{L}_{split}(\boldsymbol{c}, I_s, I_e) = \sum_{i=I_s}^{I_e} \|c_i\|_1 . \tag{3}$$

Since we divide the latent space in three parts, $\mathcal{L}_{split}$ is applied twice (see Figure 5). For the velocity part $\boldsymbol{c}_{vel}$, the indices $I_s = v+1$ and $I_e = d$ are chosen to indicate that the density part must not be used on encoding velocities, i.e., $\mathcal{L}_{split}(\boldsymbol{c}, v+1, d)$. In contrast to the previous limits, for the density part $\boldsymbol{c}_{den}$ the velocity part is indicated by choosing the indices $I_s = 0$ and $I_e = v$, i.e., $\mathcal{L}_{split}(\boldsymbol{c}, 0, v)$. First, only the velocity part $\boldsymbol{x}_{vel}$ of input $\boldsymbol{x}$ is encoded (the density part $\boldsymbol{x}_{den}$ is zero, i.e., $\bar{\boldsymbol{x}} = [\boldsymbol{x}_{vel}, 0]$), yielding $\bar{\boldsymbol{c}} = E(\bar{\boldsymbol{x}})$. Vice versa, the density part $\boldsymbol{x}_{den}$ of input $\boldsymbol{x}$ is encoded, whereas the velocity part $\boldsymbol{x}_{vel}$ is replaced with zeros, i.e., $\check{\boldsymbol{x}} = [0, \boldsymbol{x}_{den}]$, resulting in $\check{\boldsymbol{c}} = E(\check{\boldsymbol{x}})$. Therefore, the split loss is applied twice for the $\bar{\boldsymbol{c}}$ and $\check{\boldsymbol{c}}$ encodings as $\mathcal{L}_{split}(\bar{\boldsymbol{c}}, v+1, d)$ and $\mathcal{L}_{split}(\check{\boldsymbol{c}}, 0, v)$, respectively. Note that we refer to classical autoencoders trained without such a split loss as *No-Split* in the following. The split in the latent space domain is not required to separate the influence of the quantities in half. By carefully choosing the indices v and d the influence range of the individual quantities on the latent space domain can be adjusted. Later on we reference this as *split percentage*, i.e., the influence range of the velocity in the latent space domain is given as a percentage. If not stated otherwise we use a split percentage of 66%, meaning that the index v is configured so that velocity takes up 66% of the latent space.



$$\boldsymbol{c} = \left[\boldsymbol{c}_{vel} \wr \boldsymbol{c}_{den} \,|\, \boldsymbol{c}_{sp}\right], \tag{2}$$

$$\text{where} \quad \begin{aligned} \boldsymbol{c}_{vel} &= [c_0, \ldots, c_v], \\ \boldsymbol{c}_{den} &= [c_{v+1}, \ldots, c_d], \\ \boldsymbol{c}_{sp} &= [c_{d+1}, \ldots, c_{sp}]. \end{aligned}$$

Figure 4: *Visualization of a LS with $|\boldsymbol{c}| = 7$. Using the split loss, we optimally obtain a clear differentiation of the input quantities velocity and density, while their influence is spread across the whole domain using classical training methods as visualized below the No-Split label.*

In order to exhibit external control over the prediction, $\boldsymbol{c}_{sp}$ is enforced to contain parameters describing certain attributes of the simulation. While training the network, an additional soft-constraint is applied, which forces the encoder to produce the supervised parameters. The soft-constraint is implemented as the mean-squared error of the values generated by the encoder $\hat{\boldsymbol{c}}_{sp}$ and the ground truth data $\boldsymbol{c}_{sp}$ and consitutes the supervised loss $\mathcal{L}_{sup}$ as

$$\mathcal{L}_{sup}(\boldsymbol{c}_{sp}, \hat{\boldsymbol{c}}_{sp}) = \|\boldsymbol{c}_{sp} - \hat{\boldsymbol{c}}_{sp}\|_2^2 . \tag{4}$$

Additionally, an AE loss $\mathcal{L}_{AE}$ (Equation 5) is applied to the decoded field $\hat{\boldsymbol{x}}$. It forces the velocity part of the decoded field to be close to the input velocity by applying the mean-absolute error. To take the rate of change of the velocities into consideration as well, the mean-absolute error of the velocities' gradient is added to the formulation. We found empirically that the L1 norm is a good choice for our vector field data and was also applied in previous work on similar data [KCT*19]. In contrast, the density part is handled by directly applying the mean-squared error on the decoded output density and the input. For density, a scalar field, we use a L2 norm as it performed better empirically. The AE loss is thereby defined as

$$\begin{aligned} \mathcal{L}_{AE}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \lambda_{vel} &\|\boldsymbol{x}_{vel} - \hat{\boldsymbol{x}}_{vel}\|_1 \\ + \lambda_{\nabla vel} &\|\nabla \boldsymbol{x}_{vel} - \nabla \hat{\boldsymbol{x}}_{vel}\|_1 \\ + \lambda_{den} &\|\boldsymbol{x}_{den} - \hat{\boldsymbol{x}}_{den}\|_2^2 . \end{aligned} \tag{5}$$

The weights λ_{vel}, $\lambda_{\nabla vel}$ and λ_{den} put focus on the reconstruction of the velocities, their gradients or the density values. In our experiments we used $\lambda_{vel} = \lambda_{\nabla vel} = \lambda_{den} = 1$ for normalized data in accordance with the method of Kim et al. [KCT*19].

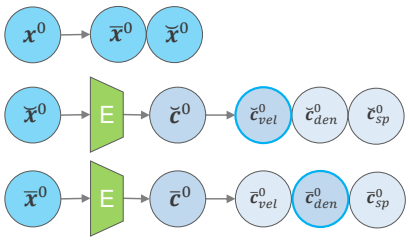


Figure 5: *The velocity and density quantities in $\boldsymbol{x}^0$ are separated, i.e., $\bar{\boldsymbol{x}}^0 = [\boldsymbol{x}^0_{vel}, 0]$ and $\check{\boldsymbol{x}}^0 = [0, \boldsymbol{x}^0_{den}]$. The LSS is enforced by applying the split loss, Equation 3, on the highlighted $\bar{\boldsymbol{c}}^0_{den}$ and $\check{\boldsymbol{c}}^0_{vel}$ for velocity and density, respectively.*

3.2. Time Prediction Network

The prediction network performs a temporal transformation of its input to the temporal consecutive state. This means a input window of w states is transformed to the consecutive state as follows $\boldsymbol{c}^0 \dots \boldsymbol{c}^{w-1} \rightarrow \tilde{\boldsymbol{c}}^w$. Therefore, the inputs to the prediction network are a series of w consecutive input states. The prediction network block contains two recurrent LSTM layers, followed by two 1D-convolutional layers. In our case of 2D smoke simulations, two consecutive latent space points of dimension 16 are used as input, i.e., $w = 2$. Those are fed to the prediction layers and result in one latent space point of dimension 16, called the residuum $\Delta\boldsymbol{c}^t$. Afterwards, the residuum is added to the last input state to arrive at the next consecutive state, i.e., $\tilde{\boldsymbol{c}}^{t+1} = \boldsymbol{c}^t + \Delta\boldsymbol{c}^t$.

Due to the subdivision capability of our autoencoder, our temporal prediction network supports external influence over the predictions it generates. After each prediction, it is possible to replace or update information without the need of re-encoding the predicted data. Instead, only parts of the predicted latent space point can be replaced, enabling fine-grained control over the flow. For example, in the case of smoke simulations, the passive smoke density quantity can be overwritten with an externally updated version, i.e., the $\boldsymbol{c}_{den}$ part is replaced by $\boldsymbol{c}$. This allows for adding new smoke sources or modifying the current flow by removing smoke from certain parts of the simulation domain.

Considering the exposure of the prediction input window w, which can be chosen freely, and the desired internal iteration count $n_i = n - w$, the additive prediction error is brought into consideration for the prediction network P while training, i.e., it is traversed n_i times. The number n defines the count of ground truth pairs given to the combined network during a training step. This leads to a combined training loss of AE and P defined as

$$\mathcal{L} = \lambda_{dir}\,\mathcal{L}_{AE,direct} + \lambda_{sup}\,\mathcal{L}_{sup} + \lambda_{sv}\,\mathcal{L}_{split,vel} + \lambda_{sd}\,\mathcal{L}_{split,den} + \lambda_p \sum_{i=0}^{n_i} \left(\mathcal{L}_{AE,pred_i}\right), \tag{6}$$

where $\mathcal{L}_{AE}$ is applied to the corresponding pairs of the decoded outputs $\tilde{\boldsymbol{x}}^2 \dots \tilde{\boldsymbol{x}}^n$ of the prediction network P and their corresponding ground-truth $\boldsymbol{x}^2 \dots \boldsymbol{x}^n$. Thereby, our final loss is the weighted sum of all the previously presented losses. The individual weights emphasize the impact of the different losses on the overall learning. In practice, we used $\lambda_{dir} = \lambda_{sup} = \lambda_{sv} = \lambda_{sd} = \lambda_p = 1$ for our examples.

In our combined training approach (see Figure 6), both networks update their weights by applying holistic gradient evaluations, i.e., are trained end-to-end. The benefit of the end-to-end training is that the spatial AE also incorporates temporal aspects when updating its weights. In addition, by recurrently applying $\mathcal{L}_{AE}$ on the predictions, the prediction network P is trained to actively minimize accumulating additive errors. To incorporate temporal awareness in the autoencoder, the decoder block is connected to the individual prediction outputs and is thereby reused several times in one network traversal. The recurrent usage of the decoding block is commonly known as weight sharing [BGL*93]. Furthermore, by applying the prediction losses on the decoded predictions, the spatial autoencoder adapts to the changes induced by the temporal prediction as well, which furthers the focus of the autoencoder to produce latent spaces suitable for temporal predictions. As a result, the prediction network is capable of robustly and accurately predicting long-term sequences of complex fluid flows.

4. Training Datasets

The datasets we used to train our networks contain randomized smoke flows simulated with an open source framework [TP18]. In total, three different scene setups were used to capture a wide range of complex physical behavior. The first scene contains a moving smoke inflow source that generates hot smoke continuously, which is rising and producing complex swirls (see Figure 7).

The second and third scenes simulate cold smoke in a cup-shaped obstacle. The former rotates the cup randomly around a fixed axis, while the latter additionally applies a translation (see Figure 7). The rising smoke and the rotating cup scene each expose one control parameter, i.e., movement on the x-axis and rotation around the z-axis, whereas the rotating and moving cup scene exposes both of these control parameters. Each of the three datasets in 2D contains 200 randomly generated scenes with 600 consecutive frames. Additionally, the moving smoke as well as the rotating and moving cup dataset was generated in 3D with 100 randomly generated scenes and 600 consecutive frames (see Table 1). The 3D moving smoke dataset contains a smoke inflow source that hovers over the ground plane and moves in x- and z-direction while generating hot smoke continuously, which is rising up and producing complex swirls. On the other hand, the 3D moving and rotating cup dataset contains scenes with a hovering cup, filled with cold smoke, that moves over the ground while also rotating around the z-axis.

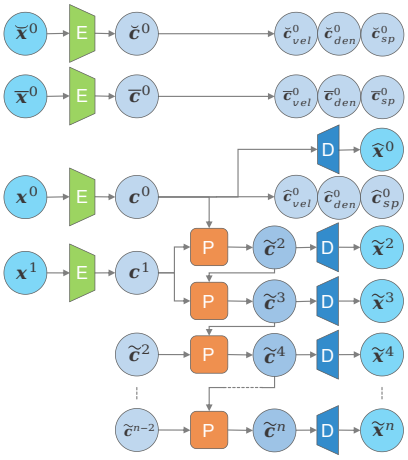


Figure 6: *Combined training of autoencoder and temporal prediction. The superscript denotes the time step of the input data. The prediction networks input window is set to $w = 2$. Thus, the count of recurrent predictions is $n_i = n - 2$. The LSS is enforced by applying the split loss, Equation 3, on $\bar{\boldsymbol{c}}^0_{den}$ and $\check{\boldsymbol{c}}^0_{vel}$ for velocity and density, respectively. A direct AE reconstruction loss, Equation 5, is only performed on $\hat{\boldsymbol{x}}^0$, whereas a prediction loss, last term of Equation 6, is performed on $\tilde{\boldsymbol{x}}^2 \dots \tilde{\boldsymbol{x}}^n$.*

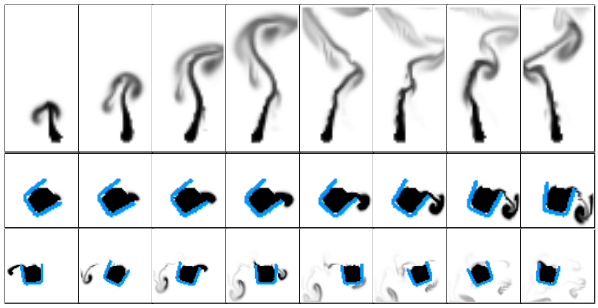


Figure 7: *Example sequences of our 2D datasets: moving smoke (top), rotating (center) and moving cup (bottom). The smoke density is shown as black and the cup-shaped obstacle in blue.*

5. Evaluation

In this section, we compare our architecture to the baseline of previous work. We also perform an ablation study on different settings of our proposed architecture to compare their respective influence on the output. We compute the mean peak signal-to-noise ratio (PSNR) for all our comparisons, i.e., larger values are better. For each case, we measure accuracy of our prediction w.r.t. density and velocity in terms of PSNR for ten simulations setups that were not seen during training.

For a thorough evaluation, we supply two prediction approaches. First, we evaluate a regular prediction approach with no reinjection of physical information (denoted *VelDen*) that is in sync with previous work [KCT*19; WBT19]. This approach is formulated as

$$\begin{aligned} \tilde{\boldsymbol{c}}^t &= P(\tilde{\boldsymbol{c}}^{t-2}, \tilde{\boldsymbol{c}}^{t-1}), \\ \tilde{\boldsymbol{x}}^t &= [\tilde{\boldsymbol{x}}^t_{vel}, \tilde{\boldsymbol{x}}^t_{den}] = D(\tilde{\boldsymbol{c}}^t), \end{aligned} \tag{7}$$

where the previously predicted latent space points $\tilde{\boldsymbol{c}}^{t-2}$ and $\tilde{\boldsymbol{c}}^{t-1}$ are used to evaluate the next time step $\tilde{\boldsymbol{c}}^t$. Afterwards, $\tilde{\boldsymbol{c}}^t$ is decoded $D(\tilde{\boldsymbol{c}}^t)$ and the density part $\tilde{\boldsymbol{x}}^t_{den}$ is directly displayed, i.e., no external physical information about the system state is influencing the output of our *VelDen* benchmarks.

In the second approach, we make use of our LSS to reinject the advected density into the prediction to benefit from well understood physical computations that keep our predictions stable and can be performed in a fast manner. The prediction process utilizing our LSS is denoted as *Vel* and is given as

$$\begin{aligned} \tilde{\boldsymbol{c}}^t &= P(\hat{\boldsymbol{c}}^{t-2}, \hat{\boldsymbol{c}}^{t-1}), \\ \tilde{\boldsymbol{x}}^t &= [\tilde{\boldsymbol{x}}^t_{vel}, \tilde{\boldsymbol{x}}^t_{den}] = D(\tilde{\boldsymbol{c}}^t), \\ \dot{\boldsymbol{x}}^t_{den} &= Adv(\dot{\boldsymbol{x}}^{t-1}_{den}, \tilde{\boldsymbol{x}}^t_{vel}), \\ \dot{\boldsymbol{c}}^t &= E([\tilde{\boldsymbol{x}}^t_{vel}, \dot{\boldsymbol{x}}^t_{den}]), \\ \hat{\boldsymbol{c}}^t &= [\tilde{\boldsymbol{c}}^t_{vel}, \dot{\boldsymbol{c}}^t_{den}], \\ \hat{\boldsymbol{x}}^t &= [\tilde{\boldsymbol{x}}^t_{vel}, \dot{\boldsymbol{x}}^t_{den}], \end{aligned} \tag{8}$$

where we are using the decoded predicted velocity $\tilde{\boldsymbol{x}}^t_{vel}$ to advect the simulation density $\dot{\boldsymbol{x}}^{t-1}_{den}$ and reinject its encoded form into our latent space $\hat{\boldsymbol{c}}^t$. The new latent space representation $\hat{\boldsymbol{c}}^t$ is thereby formed by concatenating the new encoded density $\dot{\boldsymbol{c}}^t_{den}$ and the predicted encoded velocity field $\tilde{\boldsymbol{c}}^t_{vel}$. By reinjecting the advected density field $\dot{\boldsymbol{x}}^t_{den}$, we inform the prediction network about boundary conditions as well as other known physical information that is external to the prediction state. The current simulation state is stored in $\hat{\boldsymbol{x}}^t$, i.e., the density part $\dot{\boldsymbol{x}}^t_{den}$ will be advected in the next time step whereas the encoded simulation state $\hat{\boldsymbol{c}}^t$ drives the prediction process as $\hat{\boldsymbol{c}}^{t-1}$. In the following we will ablate on different aspects of our method to evaluate their respective influence on the final results.

Table 1: *Statistics of our datasets.*

Scene Type	Resolution	# Scene	# Frames
Rotating and Moving Cup (3D)	48^3	100	600
Moving Smoke (3D)	48^3	100	600
Rotating and Moving Cup (2D)	64^2	200	600
Rotating Cup (2D)	48^2	200	600
Moving Smoke (2D)	32×64	200	600

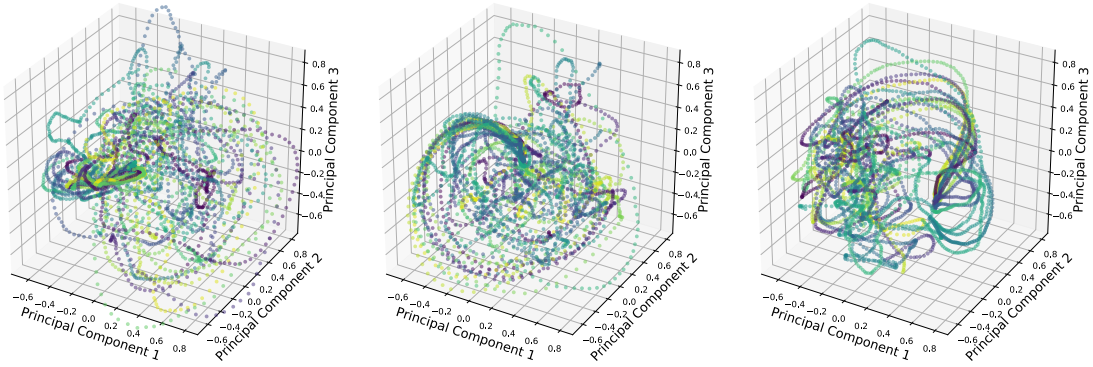
5.1. Latent Space Temporal Awareness

The temporal awareness of our spatial autoencoder is evaluated in this section, since it has a significant impact on the performance of our temporal prediction network. In Figure 8 we evaluate three networks trained with different loss functions in terms of the stability of the latent space they generate for sequences. For each of the plots, 200 frames of 20 different smoke simulations were encoded to the latent space domain with an autoencoder. The resulting latent space points were normalized with their respective maximum to the range of $[-1,1]$ and afterwards transformed to 3 dimensions with PCA. The supervised part was removed before normalization. For our comparison we chose 3 autoencoders with a latent space dimension of 16.

The results in Figure 8a were generated with a classic AE that was trained to only reproduce its input, i.e., only a direct loss on the output (Equation 5) was applied. For this classic AE no temporal constraints were imposed, and no supervised parameters were added to the latent space. The resulting PCA decomposition shows a very uneven distribution: large distances between consecutive points exist in some siutations, whereas a large part of the samples are placed closely together in a cluster.

When adding the supervised parameter loss (Equation 4) in the second evaluation [KCT*19] the trajectories become more ordered, as shown in Figure 8b, but still exhibit a noticeably uneven distribution. Thus, the supervised parameter, despite being excluded from the PCA, has a noticeable influence on the latent space construction.

In Figure 8c, the results of the AE trained with our proposed time-aware end-to-end training method are shown. This AE applies the supervised parameter loss (Equation 4) as well as the direct loss on the output (Equation 5) and was trained in combination with



(a) *No temporal constraints or supervised parameters.*

(b) *No temporal constraints but with supervised parameters.*

(c) *With temporal constraints and supervised parameters (ours).*

Figure 8: *Spatial encodings of* 200 *frames of* 20 *different smoke simulations. The latent space points are normalized to their respective maximum and processed with PCA for visualization purposes. Each color stands for a single simulation and represents a series of* 200 *frames.*

the temporal prediction network P as described in Section 3.2. The visualization of the PCA decomposition shows that a strong temporal coherence of the consecutive data points emerges. This visualization indicates why our prediction networks yield substantial improvements in terms of accuracy: the end-to-end training provides the autoencoder with gradients that guide it towards learning a latent space mapping that is suitable for temporal predictions. Intuitively, changes over time require relatively small and smooth updates, which results in the visually more regular curves shown in Figure 8c.

5.2. Simple-Split vs. Latent Space Subdivision

A simple approach to arrive at a latent space with a clear subdivision in terms of input quantities is to use two separate spatial AEs for the individual input quantities. After encoding, the two resulting latent space points $c_{vel} = E_{vel}(\boldsymbol{u})$ and $c_{den} = E_{den}(\rho)$ can be concatenated, yielding $c_{simple} = [c_{vel}, c_{den}]$. This is denoted as the *Simple-Split* variant in the following. In contrast to the simple approach, our LSS directly encodes both quantities with a single AE as $c_{LSS} = E([\boldsymbol{u}, \rho])$ and enforces the subdivision with a soft-constraint. The combined training with the prediction network is performed identical for both spatial compression versions. It becomes apparent from the results in Table 2 b), that the network trained with our soft-constraint outperforms the Simple-Split variant. Especially, when reinjecting the simulation density in the *Vel* benchmarks, we see a better PSNR value of 30.28 for $\boldsymbol{u}$ and 17.66 for ρ for our method in comparison to 26.95 and 16.35 for $\boldsymbol{u}$ and ρ, respectively. The reason for this is that the Simple-Split version can not take advantage of synergistic effects in the input data, since both input quantities are encoded in separate AEs. In contrast, our method uses the synergies of the physically interconnected velocity and density fields and robustly predicts future time steps.

Table 2: *Evaluations for Vel and VelDen predictions; rotating and moving cup: a) Internal predictions n_i; b) Simple-Split vs. LSS; c) LS dimensionality comparison*

	Vel		*VelDen*	
a) n_i	PSNR $\boldsymbol{u}$	PSNR ρ	PSNR $\boldsymbol{u}$	PSNR ρ
1	33.04	25.19	33.11	22.28
6	35.89	26.28	36.07	25.41
12	**36.61**	**26.61**	**36.61**	**25.69**
b) Type	PSNR $\boldsymbol{u}$	PSNR ρ	PSNR $\boldsymbol{u}$	PSNR ρ
Simple-Split	26.95	16.35	**29.51**	15.63
LSS (ours)	**30.28**	**17.66**	29.11	**17.12**
c) LS Dimension $\lvert c \rvert$	PSNR $\boldsymbol{u}$	PSNR ρ	PSNR $\boldsymbol{u}$	PSNR ρ
16	30.28	17.66	29.11	17.12
32	30.40	17.64	**31.58**	18.90
48	**30.86**	**17.92**	30.97	**18.96**

5.3. Internal Iterations

We compare the internal iteration count of the prediction network in the training process in Table 2 a). By performing multiple recurrent evaluations of our temporal network already in the training process we minimize the additive error build-up of many consecutive predictions. To fight the additive prediction errors over a long time horizon is important to arrive at a robust and exact predicted sequence. We chose the values of 1, 6 and 12 internal iterations for our comparison. It becomes apparent, that the network trained with 12 internal iterations and thereby the longest prediction horizon is superior in both evaluations. It should be noted, that the predictions with reinjection of the physical density field (*Vel*) have a lower error on the density than the prediction-only (*VelDen*) approach, e.g. a PSNR value of 26.61 in contrast to 25.69 for the density field of the 12 iteration version. This supports the usefulness of our pro-

Table 3: *LSS and No-Split comparison; rotating and moving cup;* 400 *time steps; split percentage of* 66%.

LS Split	Type	PSNR $\boldsymbol{u}$	PSNR ρ
No-Split	*VelDen*	17.41	10.71
LSS (ours)	*VelDen*	26.81	17.07
LSS (ours)	*Vel*	**28.15**	**21.74**

posed latent space subdivision, that is needed to reinject external information.

5.4. Latent Space Dimensionality

The latent space dimensionality has a major impact on the resulting weight count of the autoencoder as well as the complexity of the temporal predictions and thereby their difficulty. In Table 2 c) we compare latent space sizes of 16, 32 and 48. When it comes to prediction only (*VelDen*), the PSNR is better for a larger latent space dimensionality. In contrast to this observation, the PSNR value is on the same level for all latent space sizes, when the simulation density is reinjected (*Vel*). For this reason we used a latent space dimensionality of 16 for all further comparisons. Due to bouyancy, the velocity and density of our smoke simulations are loosely coupled. Thus, additional weights do not increase the overall performance when the reconstruction of the individual parts of the respective input quantities already converged.

5.5. No-Split vs. Latent Space Subdivision

In this section, we compare the performance of our LSS method with its separation of quantities in the latent space domain to classical training methods without such a separation (as for example demonstrated by Kim et al. and Wiewel et al. [KCT*19; WBT19]). Instead of making use of the split loss as defined in Equation 3, we trained a classical variant of our network, called No-Split, by removing the split loss from Equation 6 leading to the following total loss function

$$\mathcal{L} = \lambda_{dir}\,\mathcal{L}_{AE,direct} + \lambda_{sup}\,\mathcal{L}_{sup} + \lambda_p \sum_{i=0}^{n_i} \left(\mathcal{L}_{AE,pred_i}\right). \qquad (9)$$

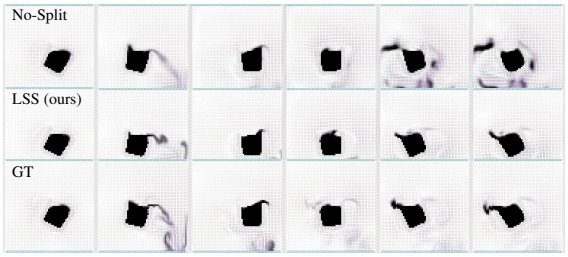


Figure 9: *Long-term prediction of* 400 *time steps: Our method robustly predicts the fluid flow, whereas the regular prediction method (No-Split) fails to capture the movement and even produces unphysical behavior. Compare the PSNR values in Table 3.*

We trained the classical version with our recurrently connected prediction network as outlined in Section 3.2. Without a split loss, both density and velocity are intertwined in the latent space. Hence, it is not possible to influence the prediction by altering the encoded density alone. Without such external correction, the prediction errors accumulate over time, which leads to a decreased prediction fidelity. Instead, with our LSS method, we enable precise injection of external physical information, leading to robust long-term predictions.

In Table 3, we compare the long-term temporal prediction of 400 simulation steps for both No-Split and our LSS network. Since No-Split autoencoders do not utilize the split loss and hence do not enable reinjection of density values, a fair evaluation can only take place for the prediction-only *VelDen* benchmark. Our LSS network clearly outperforms the No-Split version as demonstrated by the density PSNR values of 17.07 and 10.71, respectively. The resulting prediction remains stable whereas the No-Split approach fails to capture the flow throughout the prediction horizon and produces unphysical behavior, see Figure 9 for a qualitative comparison. Our method can perform even better when using the reinjection capabilities (*Vel* benchmark) considering the density PSNR value of 21.74.

5.6. Synergistic Effects of Latent Space Subdivision

As demonstrated in the previous section, our method performs better for long-term predictions than classical methods even without injecting external information. The reason for the superior performance are synergies of our latent space subdivision that we will investigate in more detail in the following.

In Figure 11, we visualize the reconstructed quantities velocity and density with our method (top left) compared to the simulated ground truth (bottom left) where our method produces quantities very close to the ground truth. Our results are generated by first encoding a single simulation frame with our autoencoder and then directly decoding the resulting latent space again as shown in Figure 10. Note that the prediction network is not used in this scheme. We then evaluate the impact of both velocity and density input ($\boldsymbol{x}_{vel}$, $\boldsymbol{x}_{den}$) and latent space subdivisions ($\boldsymbol{c}_{vel}$, $\boldsymbol{c}_{den}$) on the reconstruction quality.

Removing the density information in latent space, i.e., with $\boldsymbol{c}_{den} = 0$, the reconstructed quantities are still similar to our full method but exhibit a decreased reconstruction quality. This indi-

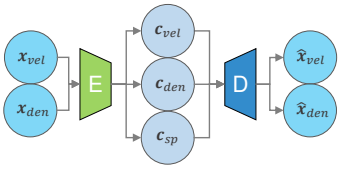


Figure 10: *Encode/Decode scheme. Our autoencoder produces the encodings $\boldsymbol{c}_{vel}$, $\boldsymbol{c}_{den}$ and $\boldsymbol{c}_{sp}$ for the inputs $\boldsymbol{x}_{vel}$ and $\boldsymbol{x}_{den}$. The fields $\hat{\boldsymbol{x}}_{vel}$ and $\hat{\boldsymbol{x}}_{den}$ are reconstructed to be close to the input. This scheme can be used to evaluate the impact of the different parts on the reconstructed fields.*

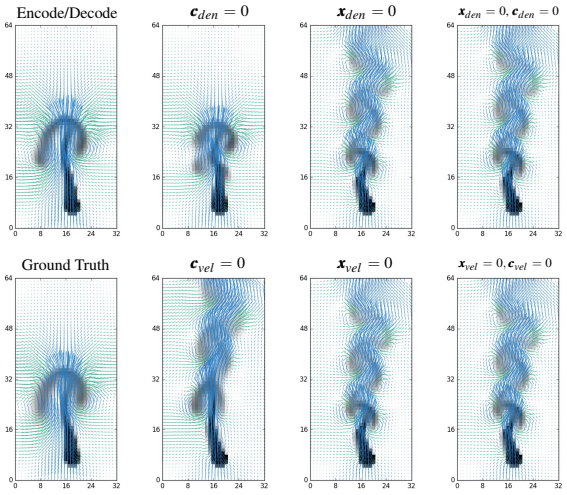


Figure 11: *Density and velocity reconstructed with our encode/decode scheme (top left) and a ground truth simulation frame (bottom left). In the following top row, we set the latent space density $\boldsymbol{c}_{den}$, the density input $\boldsymbol{x}_{den}$, or both to zero. We do the same for velocity in the bottom row. We observe that with zero input quantities ($\boldsymbol{x}_{vel} = 0$, $\boldsymbol{x}_{den} = 0$), only averaged fields are obtained for both quantities. Setting the velocity latent space to zero ($\boldsymbol{c}_{vel} = 0$) has a stronger impact on the reconstruction quality compared to setting the density latent space to zero ($\boldsymbol{c}_{den} = 0$). With the latter, the reconstructed quantities are still similar to our full method's result but exhibit a decreased reconstruction quality.*

cates that the encoded density is not essential for a meaningful reconstruction but certainly improves the quality when available. In the bottom row second left, we see that without a meaningful encoded velocity, i.e., $\boldsymbol{c}_{vel} = 0$, our reconstruction quality strongly deteriorates. This is plausible since without any velocity information, there is very little information about the state of the simulation. Velocity is vector-valued in contrast to scalar density and hence, contains more information. Furthermore, velocity encodes temporal behaviour, which is crucial for reconstructing plausible fluid motions.

If we don't provide meaningful input fields for either velocity or density ($\boldsymbol{x}_{vel} = 0$, $\boldsymbol{x}_{den} = 0$), we obtain trivially averaged velocity and density fields, as expected. Such averaged fields are close to a wide range of simulations starting at the given location. If we set the latent space quantities to zero as well ($\boldsymbol{c}_{vel} = 0$, $\boldsymbol{c}_{den} = 0$), the results are unchanged. This indicates that our split loss is indeed enforced, such that zero inputs lead to zero encoded quantities.

In summary, we find that both density and velocity fields are reconstructed holistically, i.e., both quantities are reconstructed as a whole, are aligned, and feature a similar reconstruction quality independent of which encoded quantity is set to zero. While our latent space is subdivided, both encoder and decoder provide a holistic view on the en- and decoding process. However, the encoded velocity provides a larger amount of essential information, such that the decoder is able to reconstruct both quantities even when $\boldsymbol{c}_{den} = 0$.

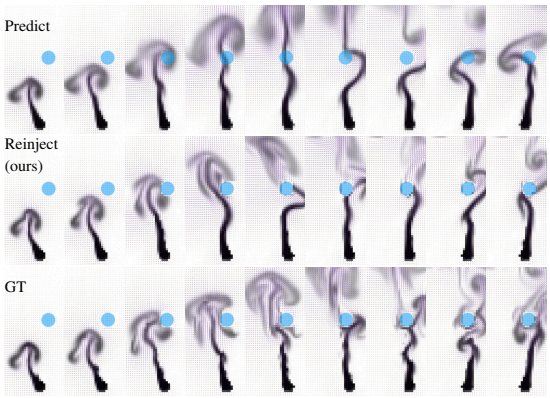


Figure 12: *A circular solid obstacle, unseen during training, is placed in the upper right. The prediction of our proposed method (center) is stable and realistic. The prediction-only approach (top) is not able to capture the obstacle.*

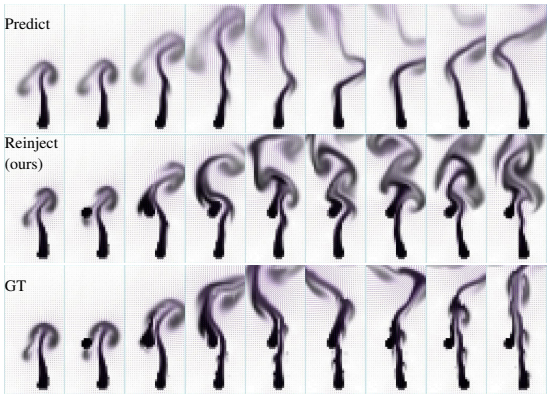


Figure 13: *An additional inflow, unseen in training, is placed during prediction. In contrast to our approach (center), the prediction only approach (top) is not able to capture the second inflow.*

6. Results

We demonstrate the effectiveness of the subdivided latent space with several generalization tests. As shown in Figure 12, our method is capable of predicting the fluid motion even when an obstacle is placed in the domain. Due to our split latent space, the obstacle can be passively injected into the prediction process by supplying an encoded density field with a masked out obstacle region. We replaced the density part of the latent space with its encoded state after advecting it with our predicted velocity field. Taking into account that the network has not encountered obstacles during training, the prediction remains stable and realistic even when majorly altering parts of the latent space encodings to include the newly placed obstacle. The prediction without injection of external information is not capturing the obstacle and thereby deviates from the ground truth. With this test we have shown that the spatial decoder distinguishes between the two input quantities and still outputs reasonable velocity fields.

In Figure 13 we show that our method is capable of predicting the fluid motion even when a new inflow region is added that was

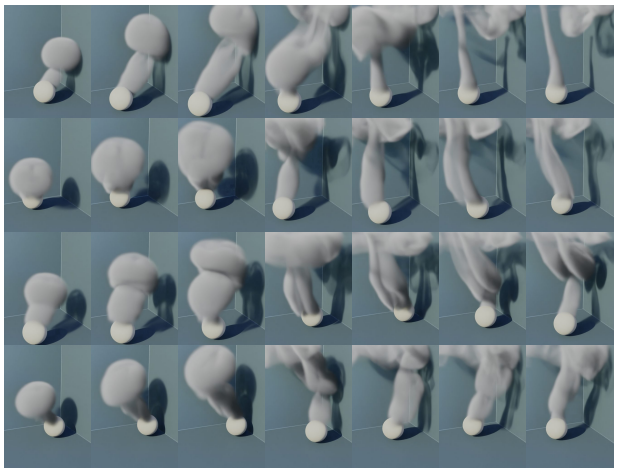


Figure 14: *Four different 3D moving smoke predictions. The movement is completely predicted by our proposed method.*

not seen while training. Our network performs reasonably well in all tested experiments due to the reinjection capabilities provided by the latent space subdivision, even though the generalization scenes were not part of the training data.

To demonstrate the capabilities of our method, we trained a 3D version of our network on the moving smoke scene; selected frames are shown in Figure 14. Additionally, we compared the runtime performance of our networks to the regular solver that was used for generating the training data (see Table 4). Even though we need to decode and encode the density field due to our reinjection method and thereby copy it from GPU to CPU memory, we still arrive at a performance measure of 0.059 seconds for an average prediction step in our 3D scene. For comparison, a traditional multi-threaded CPU-based solver takes 0.472 seconds on average for a simulation step for the same scenes.

Table 4: *Average timing of a simulation step computed via regular pressure solve and our Vel prediction scheme. While the former scales with data complexity, ours scales linearly with the domain dimension. Average of 5 scenes with 100 simulation steps each. Measured on Intel(R) Xeon(R) E5-1650 v3 (3.50GHz) and Nvidia GeForce RTX 2070.*

Scene	Resolution	Type	Solve [s]	Total [s]
Rot. mov. cup 3D	48^3	Simulation	0.891	0.960
Rot. mov. cup 3D	48^3	Prediction	**0.074**	**0.156**
Mov. smoke 3D	48^3	Simulation	0.472	0.537
Mov. smoke 3D	48^3	Prediction	**0.059**	**0.132**
Rot. mov. cup	64^2	Simulation	0.041	0.044
Rot. mov. cup	64^2	Prediction	**0.012**	**0.019**
Rot. cup	48^2	Simulation	0.018	0.019
Rot. cup	48^2	Prediction	**0.011**	**0.015**

7. Conclusion & Future Work

We have demonstrated an approach for subdividing latent spaces in a controlled manner, to improve generalization and long-term stability of physics predictions. In combination with our time-aware end-to-end training that learns a reduced representation together with the time prediction, this makes it possible to predict sequences with several hundred roll-out steps. In addition, our trained networks can be evaluated very efficiently, and yield significant speed-ups compared to traditional solvers. As future work, we believe it will be interesting to further extend the generalizing capabilities of our network such that it can cover a wider range of physical behavior. In addition, it will be interesting to explore different architectures to reduce the hardware requirements for training large 3D models with our approach.

References

[ATW15] ANDO, RYOICHI, THÜREY, NILS, and WOJTAN, CHRIS. "A Dimension-reduced Pressure Solver for Liquid Simulations". *Comput. Graph. Forum* 34.2 (May 2015), 473–480. ISSN: 0167-7055. DOI: 10.1111/cgf.12576. URL: http://dx.doi.org/10.1111/cgf.12576 2.

[BGL*93] BROMLEY, JANE, GUYON, ISABELLE, LECUN, YANN, et al. "Signature Verification Using a "Siamese" Time Delay Neural Network". *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. 1993, 737–744 5.

[Bri15] BRIDSON, ROBERT. *Fluid Simulation for Computer Graphics*. CRC Press, 2015 2, 13.

[CSK18] CUI, QIAODONG, SEN, PRADEEP, and KIM, THEODORE. "Scalable Laplacian Eigenfluids". *ACM Trans. Graph.* 37.4 (July 2018), 87:1–87:12. ISSN: 0730-0301. DOI: 10.1145/3197517.3201352. URL: http://doi.acm.org/10.1145/3197517.3201352 2.

[CT17] CHU, MENGYU and THUEREY, NILS. "Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors". *ACM Trans. Graph.* 36(4).69 (2017) 3.

[DLF12] DE WITT, TYLER, LESSIG, CHRISTIAN, and FIUME, EUGENE. "Fluid Simulation Using Laplacian Eigenfunctions". *ACM Trans. Graph.* 31.1 (Feb. 2012), 10:1–10:11. ISSN: 0730-0301. DOI: 10.1145/2077341.2077351. URL: http://doi.acm.org/10.1145/2077341.2077351 2.

[GN07] GUPTA, MOHIT and NARASIMHAN, SRINIVASA G. "Legendre Fluids: A Unified Framework for Analytic Reduced Space Modeling and Rendering of Participating Media". *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '07. San Diego, California: Eurographics Association, 2007, 17–25. ISBN: 978-1-59593-624-0. URL: http://dl.acm.org/citation.cfm?id=1272690.1272693 2.

[JSK16] JONES, AARON DEMBY, SEN, PRADEEP, and KIM, THEODORE. "Compressing fluid subspaces". *Symposium on Computer Animation*. ACM/Eurographics. 2016, 77–84 2.

[KCT*19] KIM, BYUNGSOO, C. AZEVEDO, VINICIUS, THUEREY, NILS, et al. "Deep Fluids: A Generative Network for Parameterized Fluid Simulations". *Computer Graphics Forum (Proc. Eurographics)* 38.2 (2019) 2–4, 6, 8.

[KD13] KIM, THEODORE and DELANEY, JOHN. "Subspace Fluid Resimulation". *ACM Trans. Graph.* 32.4 (July 2013), 62:1–62:9. ISSN: 0730-0301. DOI: 10.1145/2461912.2461987. URL: http://doi.acm.org/10.1145/2461912.2461987 2.

[LJS*15] LADICKÝ, L'UBOR, JEONG, SOHYEON, SOLENTHALER, BARBARA, et al. "Data-driven fluid simulations using regression forests". *ACM Transactions on Graphics* 34.6 (Oct. 2015), 1–9 1, 2.

[LMH*15] LIU, BEIBEI, MASON, GEMMA, HODGSON, JULIAN, et al. "Model-reduced Variational Fluid Simulation". *ACM Trans. Graph.* 34.6 (Oct. 2015), 244:1–244:12. ISSN: 0730-0301. DOI: 10.1145/2816795.2818130. URL: http://doi.acm.org/10.1145/2816795.2818130 2.

[LR09] LONG, BENJAMIN and REINHARD, ERIK. "Real-time Fluid Simulation Using Discrete Sine/Cosine Transforms". *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. Boston, Massachusetts: ACM, 2009, 99–106. ISBN: 978-1-60558-429-4. DOI: 10.1145/1507149.1507165. URL: http://doi.acm.org/10.1145/1507149.1507165 2.

[MJKW18] MORTON, JEREMY, JAMESON, ANTONY, KOCHENDERFER, MYKEL J, and WITHERDEN, FREDDIE. "Deep dynamical modeling and control of unsteady fluid flows". *Proceedings of Neural Information Processing Systems*. 2018 2, 3.

[RMC16] RADFORD, ALEC, METZ, LUKE, and CHINTALA, SOUMITH. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". *Proc. ICLR* (2016) 2.

[RWTT14] RAVEENDRAN, KARTHIK, WOJTAN, CHRIS, THUEREY, NILS, and TURK, GREG. "Blending Liquids". *ACM Trans. Graph.* 33.4 (July 2014), 137:1–137:10. ISSN: 0730-0301. DOI: 10.1145/2601097.2601126. URL: http://doi.acm.org/10.1145/2601097.2601126 2.

[SDN18] SATO, SYUHEI, DOBASHI, YOSHINORI, and NISHITA, TOMOYUKI. "Editing Fluid Animation Using Flow Interpolation". *ACM Trans. Graph.* 37.5 (Sept. 2018), 173:1–173:12. ISSN: 0730-0301. DOI: 10.1145/3213771. URL: http://doi.acm.org/10.1145/3213771 2.

[SFK*08] SELLE, ANDREW, FEDKIW, RONALD, KIM, BYUNGMOON, et al. "An Unconditionally Stable MacCormack Method". *J. Sci. Comput.* 35.2-3 (June 2008), 350–371 2.

[Thu16] THUEREY, NILS. "Interpolations of Smoke and Liquid Simulations". *ACM Trans. Graph.* 36.1 (Sept. 2016), 3:1–3:16. ISSN: 0730-0301. DOI: 10.1145/2956233. URL: http://doi.acm.org/10.1145/2956233 2.

[TLP06] TREUILLE, ADRIEN, LEWIS, ANDREW, and POPOVIĆ, ZORAN. "Model Reduction for Real-time Fluids". *ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. Boston, Massachusetts: ACM, 2006, 826–834. ISBN: 1-59593-364-6. DOI: 10.1145/1179352.1141962. URL: http://doi.acm.org/10.1145/1179352.1141962 2.

[TP18] THUEREY, NILS and PFAFF, TOBIAS. *MantaFlow*. 2018 5.

[TSSP17] TOMPSON, JONATHAN, SCHLACHTER, KRISTOFER, SPRECHMANN, PABLO, and PERLIN, KEN. "Accelerating eulerian fluid simulation with convolutional networks". *Proceedings of the 34th ICML Vol. 70*. JMLR. org. 2017, 3424–3433 3.

[UHT18] UM, KIWON, HU, XIANGYU, and THUEREY, NILS. "Liquid splash modeling with neural networks". *Computer Graphics Forum*. Vol. 37. 8. Wiley Online Library. 2018, 171–182 3.

[WBT19] WIEWEL, STEFFEN, BECHER, MORITZ, and THUEREY, NILS. "Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow". *Computer Graphics Forum* 38.2 (2019), 71–82 2, 6, 8.

[WST09] WICKE, MARTIN, STANTON, MATTHEW, and TREUILLE, ADRIEN. "Modular Bases for Fluid Dynamics". *ACM Trans. Graph.* 28.3 (Aug. 2009), 39 2.

[WZX*16] WU, JIAJUN, ZHANG, CHENGKAI, XUE, TIANFAN, et al. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling". *Advances in Neural Information Processing Systems*. 2016, 82–90 2.

[XFCT18] XIE, YOU, FRANZ, ERIK, CHU, MENGYU, and THUEREY, NILS. "tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow". *SIGGRAPH* (2018) 3.

[YYX16] YANG, CHENG, YANG, XUBO, and XIAO, XIANGYUN. "Data-driven projection method in fluid simulation". *Computer Animation and Virtual Worlds* 27.3-4 (May 2016), 415–424. ISSN: 15464261. DOI: 10.1002/cav.1695. URL: http://doi.wiley.com/10.1002/cav.1695 3.

Supplemental Document for Latent Space Subdivision: Stable and Controllable Time Predictions for Fluid Flow

Appendix A: Evaluation

Prediction Window Size

The prediction window w describes the count of consecutive time steps that are taken as input by the temporal prediction network. In our comparison we tested window sizes ranging from 2 over 3 up to 4 consecutive input steps. The results in terms of PSNR values are displayed in Table 5 and Table 6.

Table 5: *Prediction window w comparison Vel*

w	PSNR $\boldsymbol{u}$	PSNR ρ
4	29.67	17.04
3	29.79	16.87
2	**30.28**	**17.66**

Table 6: *Prediction window w comparison VelDen*

w	PSNR $\boldsymbol{u}$	PSNR ρ
4	**29.98**	**18.24**
3	29.52	17.84
2	29.11	17.12

It becomes apparent that the prediction-only approach (*VelDen*) benefits from a larger input window, whereas the *Vel* approach with reinjected external information performs best with a smaller input window.

Latent Space Split Percentage

We evaluated the impact of the latent space split percentage on three of our datasets. Therefore, we trained multiple models with different split percentages on the individual datasets. The comparison for our moving smoke scene is shown in Table 7 and Table 8. The latter are the results of the prediction-only evaluation (denoted *VelDen*), whereas the first table presents the results of our reinjected density approach (denoted *Vel*). In this experiment all split versions are outperformed by the No-Split version in the prediction-only setup with PSNR values of 29.71 and 18.03 for velocity and density, respectively.

Table 7: *LS split comparison Vel; moving smoke*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
0.33	28.07	15.84
0.5	29.28	16.49
0.66	**30.28**	**17.66**

Table 8: *LS split comparison VelDen; moving smoke*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
No-Split	**29.71**	**18.03**
0.33	28.49	16.63
0.5	29.06	17.38
0.66	29.11	17.12

Table 9: *LS split comparison Vel; rotating cup*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
0.33	36.67	28.46
0.5	36.66	29.22
0.66	**38.52**	**29.73**

Table 10: *LS split comparison VelDen; rotating cup*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
No-Split	**37.90**	22.68
0.33	37.52	25.01
0.5	36.77	25.13
0.66	37.57	**25.32**

Table 11: *LS split comparison Vel; rotating and moving cup*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
0.33	35.67	25.10
0.5	**36.94**	**26.59**
0.66	36.50	26.25

Table 12: *LS split comparison VelDen; rotating and moving cup*

LS Split	PSNR $\boldsymbol{u}$	PSNR ρ
0.33	**37.89**	**26.45**
0.5	37.30	26.16
0.66	37.56	26.14

Table 13: *No-Split and LSS comparison; rotating cup; 100 time steps*

LS Split	Type	PSNR $\boldsymbol{u}$	PSNR ρ
No-Split	*VelDen*	37.90	22.68
0.66 (ours)	*VelDen*	37.57	25.32
0.66 (ours)	*Vel*	**38.52**	**29.73**

In contrast, the networks trained on the rotating cup dataset behave different as shown in Table 9 and Table 10. The classic No-Split version is outperformed by all other split versions in terms of density PSNR values in the prediction-only (*VelDen*) setup. In the reinjected density evaluation (*Vel*), the benefit of latent space splitting becomes even more apparent when comparing the PSNR values of velocity, 38.52 and density, 29.73 of the 0.66 network with the velocity PSNR of 37.90 and density PSNR 22.68 of the No-Split version.

No-Split vs. Latent Space Subdivision

In this section we present additional results for our rotating cup dataset. See the main document for a long-term comparison of No-Split vs. LSS for our more complicated moving and rotating cup dataset. In Table 13 we compare the temporal prediction performance of a No-Split version and our 0.66 LSS version over a time horizon of 100 simulation steps. Our LSS 0.66 version with a density PSNR value of 29.73 clearly outperforms the No-Split version with a density PSNR value of 22.68.

Generalization

Additionally, we show in Figure 15 that our method recovers from the removal of smoke in a certain sink-region and is capable of predicting the fluid motion.

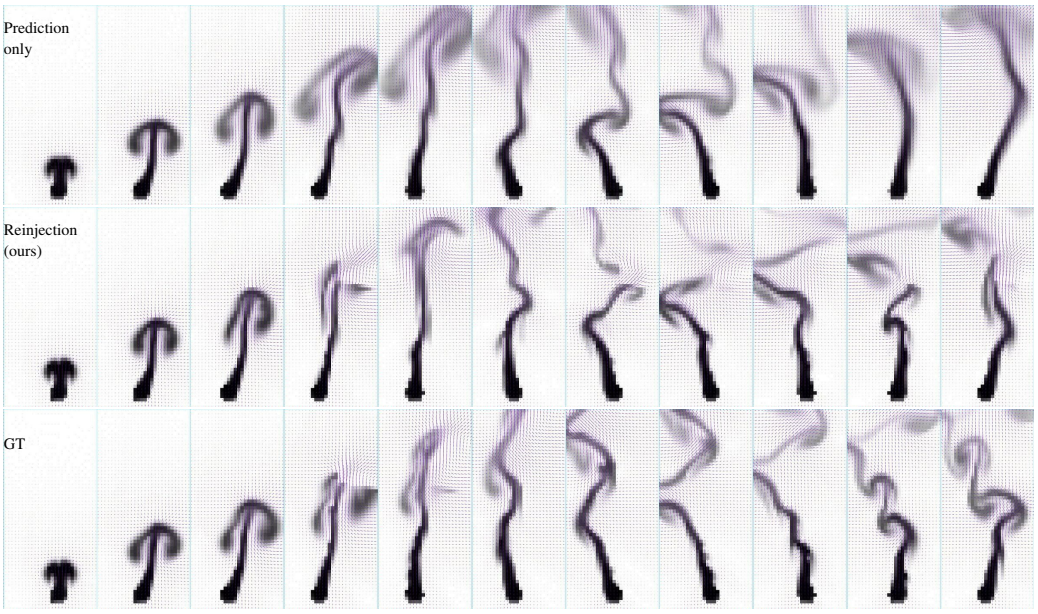


Figure 15: *An sink is placed in the upper right of our moving smoke scene. This was unseen during training. The prediction by our proposed method remains stable and realistic. In the second row density reinjection was applied. In the top row no external information was injected. Thus, the sink can't be processed by the network.*

Appendix B: Fluid Flow Data

Our work concentrates on single-phase flows, modelled by a pressure-velocity formulation of the incompressible Navier-Stokes equations as highlighted in Section 2. Thereby, we apply a classic NS solver to simulate our smoke flows based on R. Bridson [Bri15]. In addition to Section 4, more information about the simulation procedure is provided in the following.

Simulation Setup

The linear system for pressure projection is solved with a conjugate gradient method. The conjugate gradient (CG) solver accuracy is set to $1 \cdot 10^{-4}$ for our moving smoke dataset, whereas an accuracy of $1 \cdot 10^{-3}$ is utilized for the moving cup datasets. We generated all our datasets with a time step of 0.5. Depending on the behavioral requirements of our different experiments with rising, hot and sinking, cold smoke we use the Boussinesq model with the smoke density in combination with a gravity constant of $(0.0, -4 \cdot 10^{-3}, 0.0)$ for the moving and rising smoke and $(0.0, 1 \cdot 10^{-3}, 0.0)$ for the rotating cup dataset. To arrive at a more turbulent flow behavior, the gravity constant was set to $(0.0, 1 \cdot 10^{-2}, 0.0)$ for our moving and rotating cup dataset. We do not apply other forces or additional viscosity. We purely rely on numerical diffusion to introduce viscosity effects.

In combination with the quantities required by our classic NS setup, namely flow velocity $\boldsymbol{u}$, pressure p and density ρ, we also need a flag grid f, an obstacle velocity field $\boldsymbol{u}_{obs}$ and the corresponding obstacle levelset for our obstacle supporting scenes. Thereby our density ρ is passively advected within the flow velocity $\boldsymbol{u}$.

To handle the obstacle movement accordingly, we calculate the obstacle velocity field by evaluating the displacement per mesh vertex of the previous to the current time step and applying the interpolated velocities to the according grid cells of the obstacle velocity field. Afterwards, the obstacle velocity field values are averaged to represent a correct discretization.

In Algorithm 1 the simulation procedure of the moving smoke dataset is shown. For our obstacle datasets the procedure in Algorithm 2 is used, with the prediction algorithm given in Algorithm 3. Boundary conditions are abbreviated with BC in these algorithm.

Algorithm 1 Moving smoke simulation

while $t \rightarrow t+1$ **do**
  $\rho \leftarrow$ applyInflowSource(ρ, s)
  $\rho \leftarrow$ advect(ρ, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ advect($\boldsymbol{u}$, $\boldsymbol{u}$)
  $f \leftarrow$ setWallBCs(f, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ addBuoyancy(ρ, $\boldsymbol{u}$, f, $\boldsymbol{g}$)
  $p \leftarrow$ solvePressure(f, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ correctVelocity($\boldsymbol{u}$, p)
end while

Training Datasets

In Figure 16, Figure 17 and Figure 18 multiple simulations contained in our training data set are displayed.

Algorithm 2 Rotating and moving cup

while $t \rightarrow t+1$ **do**
  $\rho \leftarrow$ applyInflowSource(ρ, s)
  $\rho \leftarrow$ advect(ρ, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ advect($\boldsymbol{u}$, $\boldsymbol{u}$)
  $\boldsymbol{u}_{obs} \leftarrow$ computeObstacleVelocity($obstacle^t$, $obstacle^{t+1}$)
  $f \leftarrow$ setObstacleFlags($obstacle^t$)
  $f \leftarrow$ setWallBCs(f, $\boldsymbol{u}$, $obstacle^t$, $\boldsymbol{u}_{obs}$)
  $\boldsymbol{u} \leftarrow$ addBuoyancy(ρ, $\boldsymbol{u}$, f, $\boldsymbol{g}$)
  $p \leftarrow$ solvePressure(f, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ correctVelocity($\boldsymbol{u}$, p)
end while

Algorithm 3 Rotating and moving cup network prediction *Vel*

while $t \rightarrow t+1$ **do**
  $\rho \leftarrow$ applyInflowSource(ρ, s)
  $\rho \leftarrow$ advect(ρ, $\boldsymbol{u}$)
  $\boldsymbol{u} \leftarrow$ advect($\boldsymbol{u}$, $\boldsymbol{u}$)
  $\boldsymbol{u}_{obs} \leftarrow$ computeObstacleVelocity($obstacle^t$, $obstacle^{t+1}$)
  $f \leftarrow$ setObstacleFlags($obstacle^t$)
  $f \leftarrow$ setWallBCs(f, $\boldsymbol{u}$, $obstacle^t$, $\boldsymbol{u}_{obs}$)
  $\dot{\boldsymbol{c}}^t \leftarrow$ encode($\tilde{\boldsymbol{u}}^t$, ρ^t)
  $\hat{\boldsymbol{c}}^t \leftarrow [\tilde{\boldsymbol{c}}^t_{vel}, \dot{\boldsymbol{c}}^t_{den}]$
  $\tilde{\boldsymbol{c}}^{t+1} \leftarrow$ predict($\hat{\boldsymbol{c}}^{t-1}$, $\hat{\boldsymbol{c}}^t$)
  $\tilde{\boldsymbol{u}}^{t+1}, \tilde{\rho}^{t+1} \leftarrow$ decode($\tilde{\boldsymbol{c}}^{t+1}$) ▷ $\tilde{\rho}^{t+1}$ is not used
  $\boldsymbol{u}^{t+1} \leftarrow \tilde{\boldsymbol{u}}^{t+1}$ ▷ overwrite the velocity with the prediction
end while

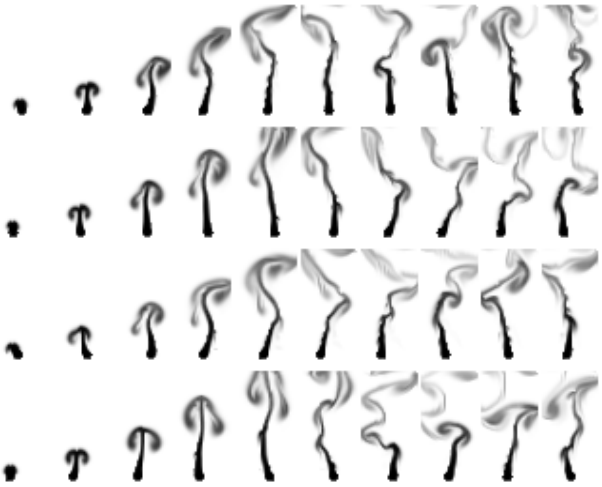


Figure 16: *Four example sequences of our moving smoke dataset. For visualization purposes we display frames* 20 *to* 200 *with a step size of* 20 *for the respective scenes. The smoke density is shown as black.*

Figure 17: *Three example sequences of our rotating cup dataset. For visualization purposes we display frames* 40 *to* 180 *with a step size of* 20 *for the respective scenes. The cup-shaped obstacle is highlighted in blue, whereas the smoke density is shown as black.*

Figure 18: *Three example sequences of our rotating and moving cup dataset. For visualization purposes we display frames* 40 *to* 180 *with a step size of* 20 *for the respective scenes. The cup-shaped obstacle is highlighted in blue, whereas the smoke density is shown as black.*