EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)

Volume 38 (2019), Number 2

Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow

S. Wiewel[1], M. Becher[1], N. Thuerey[1] †

[1]Technical University of Munich

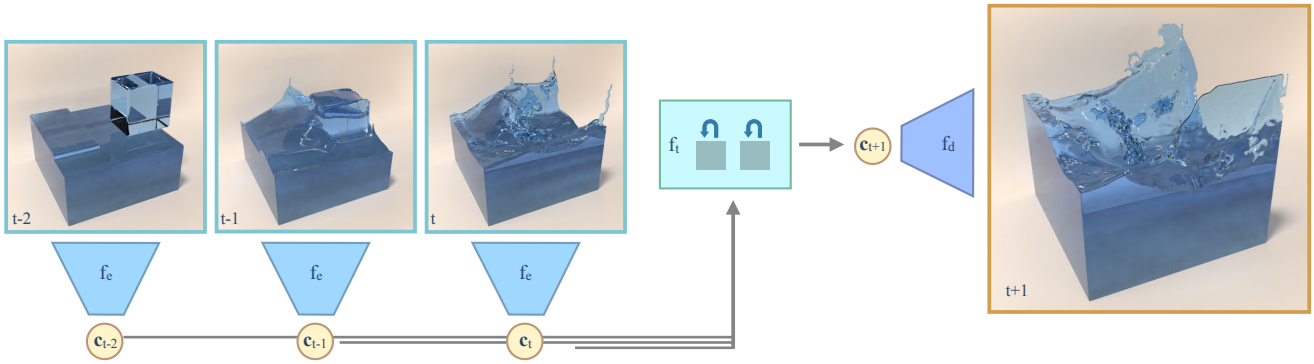


Figure 1: *Our method encodes multiple steps of a simulation field, typically pressure, into a reduced latent representation with a convolutional neural network. A second neural network with LSTM units then predicts the latent space code for one or more future time steps, yielding large reductions in runtime compared to regular solvers.*

Abstract

We propose a method for the data-driven inference of temporal evolutions of physical functions with deep learning. More specifically, we target fluid flows, i.e. Navier-Stokes problems, and we propose a novel LSTM-based approach to predict the changes of pressure fields over time. The central challenge in this context is the high dimensionality of Eulerian space-time data sets. We demonstrate for the first time that dense 3D+time functions of physics system can be predicted within the latent spaces of neural networks, and we arrive at a neural-network based simulation algorithm with significant practical speed-ups. We highlight the capabilities of our method with a series of complex liquid simulations, and with a set of single-phase buoyancy simulations. With a set of trained networks, our method is more than two orders of magnitudes faster than a traditional pressure solver. Additionally, we present and discuss a series of detailed evaluations for the different components of our algorithm.

CCS Concepts

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

1. Introduction

The variables we use to describe real world physical systems typically take the form of complex functions with high dimensionality. Especially for transient numerical simulations, we usually employ continuous models to describe how these functions evolve over time. For such models, the field of computational methods has been highly successful at developing powerful numerical algorithms that accurately and efficiently predict how the natural phenomena under consideration will behave. In the following, we take a different view on this problem: instead of relying on analytic expressions, we use a deep learning approach to infer physical functions based on data. More specifically, we will focus on the temporal evolution of complex functions that arise in the context of fluid flows. Fluids encompass a large and important class of materials in human environments, and as such they're particularly interesting candidates for learning models.

While other works have demonstrated that machine learning methods are highly competitive alternatives to traditional meth-

† This work was funded by the ERC Starting Grant *realFlow* (StG-2015-637014).

ods, e.g., for computing local interactions of particle based liquids [LJS*15], to perform divergence free projections for a single point in time [TSSP16], or for adversarial training of high resolution flows [XFCT18], few works exist that target temporal evolutions of physical systems. While first works have considered predictions of Lagrangian objects such as rigid bodies [WZW*17], and control of two dimensional interactions [MTP*18], the question whether neural networks (NNs) can predict the evolution of complex three-dimensional functions such as pressure fields of fluids has not previously been addressed. We believe that this is a particularly interesting challenge, as it not only can lead to faster forward simulations, as we will demonstrate below, but also could be useful for giving NNs predictive capabilities for complex inverse problems.

The complexity of nature at human scales makes it necessary to finely discretize both space and time for traditional numerical methods, in turn leading to a large number of degrees of freedom. Key to our method is reducing the dimensionality of the problem using *convolutional neural networks* (CNNs) with respect to both time and space. Our method first learns to map the original, three-dimensional problem into a much smaller spatial *latent space*, at the same time learning the inverse mapping. We then train a second network that maps a collection of reduced representations into an encoding of the temporal evolution. This reduced temporal state is then used to output a sequence of spatial latent space representations, which are decoded to yield the full spatial data set for a point in time. A key advantage of CNNs in this context is that they give us a natural way to compute accurate and highly efficient nonlinear representations. We will later on demonstrate that the setup for computing this reduced representation strongly influences how well the time network can predict changes over time, and we will demonstrate the generality of our approach with several liquid and single-phase problems. The specific contributions of this work are:

- a first LSTM architecture to predict temporal evolutions of dense, physical 3D functions in learned latent spaces,
- an efficient encoder and decoder architecture, which by means of a strong compression, yields a very fast simulation algorithm,
- in addition to a detailed evaluation of training modalities.

2. Related Work and Background

Despite being a research topic for a long time [RHW88], the interest in neural network algorithms is a relatively new phenomenon, triggered by seminal works such as *ImageNet* [KSH12]. In computer graphics, such approaches have led to impressive results, e.g., for synthesizing novel viewpoints of natural scenes [FNPS16], to generate photorealistic face textures [SWH*16], and to robustly transfer image styles between photographs [LPSB17], to name just a few examples. The underlying optimization approximates an unknown function $f^*(x) = y$, by minimizing an associated loss function L such that $f(x,\theta) \approx y$. Here, θ denotes the degrees of freedom of the chosen representation for f. For our algorithm, we will consider deep neural networks. With the right choice of L, e.g., an L_2 norm in the simplest case, such a neural network will approximate the original function f^* as closely as possible given its internal structure. A single layer l of an NN can be written as $a^l = \sigma(W_l a^{l-1} + b_l)$, where a^i is the output of the i'th layer, σ represents an activation function, and W_l, b_l denote weight matrix and bias, respectively. In the following, we collect the weights W_l, b_l for all layers l in θ.

The latent spaces of generative NNs were shown to be powerful tools in image processing and synthesis [RMC16, WZX*16]. They provide a non-linear representation that is closely tied to the given data distribution, and our approach leverages such a latent space to predict the evolution of dense physical functions. While others have demonstrated that trained feature spaces likewise pose very suitable environments for high-level operations with impressive results [UGB*16], we will focus on latent spaces of autoencoder networks in the following. The sequence-to-sequence methods which we will use for time prediction have so far predominantly found applications in the area of natural language processing, e.g., for tasks such as machine translation [SVL14]. These recurrent networks are especially popular for control tasks in reinforcement learning environments [MBM*16]. Recently, impressive results were also achieved for tasks such as automatic video captioning [XYZM17].

Using neural networks in the context of physics problems is a new area within the field of deep learning methods. Several works have targeted predictions of Lagrangian objects based on image data. E.g., Battaglia et al. [BPL*16] predict two-dimensional physics, a method that was subsequently extended to videos [WZW*17]. Another line of work proposed a specialized architecture for two-dimensional rigid bodies physics [CUTT16], while others have targeted predictions of liquid motions for robotic control [SF17] Farimani et al. [FGP17] proposed an adversarial training approach to infer solutions for two-dimensional physics problems, such as heat diffusion and lid driven cavity flows. Other researchers have proposed networks to learn PDEs [LLMD17] by encoding the unknown differential operators with convolutions. To model the evolution of a subsurface multiphase flow others are using proper orthogonal decomposition in combination with recurrent neural networks (RNNs) [KE18]. In the field of weather prediction, short-term prediction architectures evolved that internally also make use of RNN layers [SCW*15]. Other works create velocity field predictions by learning parameters for Reynolds Averaged Navier Stokes models from simulation data [LKT16]. In addition, learned Koopman operators [MWJK18] were proposed for representing temporal changes, whereas Lusch et al. are searching for representations of Koopman eigenfunctions to globally linearize dynamical systems using deep learning [LKB18]. While most of these works share our goal to infer Eulerian functions for physical models, they are limited to relatively simple, two dimensional problems. In contrast, we will demonstrate that our reduced latent space representation can work with complex functions with up to several million degrees of freedom.

We focus on flow physics, for which we employ the well established *Navier-Stokes* (NS) model. Its incompressible form is given by

$$\partial \mathbf{u}/\partial t + \mathbf{u} \cdot \nabla \mathbf{u} = -1/\rho \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g}\,, \quad \nabla \cdot \mathbf{u} = 0, \qquad (1)$$

where the most important quantities are flow velocity $\mathbf{u}$ and pressure p. The other parameters $\rho, \nu, \mathbf{g}$ denote density, kinematic viscosity and external forces, respectively. For liquids, we will assume

that a signed-distance function ϕ is either advected with the flow, or reconstructed from a set of advected particles.

In the area of visual effects, Kass and Miller were the first to employ height field models [KM90], while Foster and Metaxas employed a first three-dimensional NS solver [FM96]. After Jos Stam proposed an unconditionally stable advection and time integration scheme [Sta99], it led to powerful liquid solvers based on the particle levelset [FF01], in conjunction with accurate free surface boundary conditions [ENGF03]. Since then, the *fluid implicit particle* (FLIP) method has been especially popular for detailed liquid simulations [ZB05], and we will use it to generate our training data. Solvers based on these algorithms have subsequently been extended with accurate boundary handling [BBB07] synthetic turbulence [KTJG08], or narrow band algorithms [FAW*16], to name just a few examples. A good overview of fluid simulations for computer animation can be found in the book by R. Bridson [Bri15]. Aiming for more complex systems, other works have targeted coupled reduced order models, or sub-grid coupling effects [TLK16, FMB*17]. While we do not target coupled fluid solvers in our work, these directions of course represent interesting future topics.

Beyond these primarily grid-based techniques, *smoothed particle hydrodynamics* (SPH) are a popular Lagrangian alternative [MCG03, MMCK14]. However, we will focus on Eulerian solvers in the following, as CNNs are particularly amenable to grid-based discretizations. The pressure function has received special attention, as the underlying iterative solver is often the most expensive component in an algorithm. E.g., techniques for dimensionality reduction [LZF10, ATW15], and fast solvers [MST10, ICS*14] have been proposed to diminish its runtime impact.

In the context of fluid simulations and machine learning for animation, a regression forest based approach for learning SPH interactions has been proposed by Ladicky et al. [LJS*15]. Other graphics works have targeted learning flow descriptors with CNNs [CT17], or learning the statistics of splash formation [UHT17], and two-dimensional control problems [MTP*18]. While pressure projection algorithms with CNNs [TSSP16, YYX16] shares similarities with our work on first sight, they are largely orthogonal. Instead of targeting divergence freeness for a single instance in time, our work aims for learning its temporal evolution over the course of many time steps. An important difference is also that the CNN-based projection so far has only been demonstrated for smoke simulations, similar to other model-reduced simulation approaches [TLP06]. For all of these methods, handling the strongly varying free surface boundary conditions remains an open challenge, and we demonstrate below that our approach works especially well for liquids.

3. Method

The central goal of our method is to predict future states of a physical function $\mathbf{x}$. While previous works often consider low dimensional Lagrangian states such as center of mass positions, $\mathbf{x}$ takes the form of a dense Eulerian function in our setting. E.g., it can represent a spatio-temporal pressure function, or a velocity field. Thus, we consider $\mathbf{x} : \mathbb{R}^3 \times \mathbb{R}^+ \rightarrow \mathbb{R}^d$, with $d = 1$ for scalar functions such as pressure, and $d = 3$ for vectorial functions such as velocity fields. As a consequence, $\mathbf{x}$ has very high dimensionality when discretized, typically on the order of millions of spatial degrees of freedom.

Given a set of parameters θ and a functional representation f_t our goal is to predict the o future states $\mathbf{x}(t+h)$ to $\mathbf{x}(t+oh)$ as closely as possible given a current state and a series of n previous states, i.e.,

$$f_t\big(\mathbf{x}(t-nh),...,\mathbf{x}(t-h),\mathbf{x}(t)\big) \approx \big[\mathbf{x}(t+h),\ldots,\mathbf{x}(t+oh)\big]. \quad (2)$$

To provide better visual clarity the set of weights, i.e. learnable parameters, θ is omitted in the function definitions.

Due to the high dimensionality of $\mathbf{x}$, directly solving Eq. (2) would be very costly. Thus, we employ two additional functions f_d and f_e, that compute a low dimensional encoding. The encoder f_e maps into an m_s dimensional space $\mathbf{c} \in \mathbb{R}^{m_s}$ with $f_e(\mathbf{x}(t)) = \mathbf{c}^t$, whereas the decoding function f_d reverses the mapping with $f_d(\mathbf{c}^t) = \mathbf{x}(t)$. Thus, f_d and f_e here represent spatial decoder and encoder models, respectively. Given such an en- and decoder, we rephrase the problem above as

$$\begin{aligned} \tilde{f}_t\big(f_e(\mathbf{x}(t-nh)),...,f_e(\mathbf{x}(t))\big) &\approx \big[\mathbf{c}^{t+h},\ldots,\mathbf{c}^{t+oh}\big] \\ f_d\big(\big[\mathbf{c}^{t+h},\ldots,\mathbf{c}^{t+oh}\big]\big) &\approx \big[\mathbf{x}(t+h),\ldots,\mathbf{x}(t+oh)\big] \\ f_d\big(\tilde{f}_t(f_e(\mathbf{x}(t-nh)),...,f_e(\mathbf{x}(t)))\big) &\approx f_t(\mathbf{x}(t-nh),...,\mathbf{x}(t)) \end{aligned} \quad (3)$$

We will use CNNs for f_d and f_e, and thus the space $\mathbf{c}$ is given by their learned *latent space*. We choose its dimensionality m_s such that the temporal prediction problem above becomes feasible for dense three dimensional samples.

Our prediction network that models the function $\tilde{f}_t$ will likewise employ an encoder-decoder structure, and turns the temporal stack of encoded data points $f_e(\mathbf{x})$ into a reduced representation $\mathbf{d}$, which we will refer to as *temporal context* below. The architectures of the spatial and temporal networks are described in the following sections Sec. 3.1 and Sec. 3.2, respectively.

3.1. Reducing Dimensionality

In order to reduce the spatial dimensionality of our inference problem, we employ a fully convolutional autoencoder architecture [MMCS11]. Our autoencoder (AE) consists of the aforementioned encoding and decoding functions f_e, f_d and is trained to reconstruct the quantity $\mathbf{x}$ as accurately as possible w.r.t. an L_2 norm, i.e.

$$\min_{\theta_d,\theta_e} |f_d(f_e(\mathbf{x}(t))) - \mathbf{x}(t)|_2, \quad (4)$$

where θ_d, θ_e represent the parameters of the decoder and encoder, respectively. We use a series of convolutional layers activated by leaky rectified linear units (LeakyReLU) [MHN13] for encoder and decoder, with a bottleneck layer of dimensionality m_s. This layer yields the latent space encoding that we use to predict the temporal evolution. Both encoder and decoder consist of 6 convolutional layers that increase / decrease the number of features by a factor of 2. In total, this yields a reduction factor of 256.

In the following, we will explain additional details of the autoencoder pre-training, and layer setup. We denote layers in the encoder and decoder stack as f_{e_i} and f_{d_j}, where $i, j \in [0, l]$, with i, j being

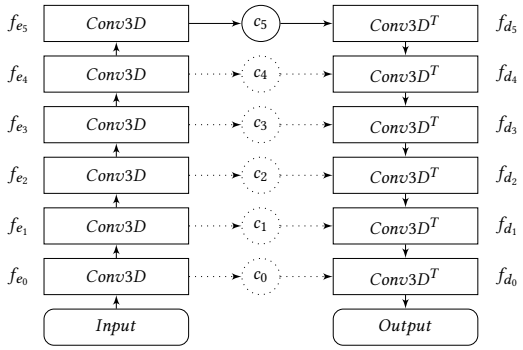


Figure 2: *Overview of the autoencoder network architecture See Table 1 for a detailed description of the layers. Note that the paths from f_{e_k} over c_k to f_{d_k} are only active in pretraining stage k. After pretraining only the path f_{e_5} over c_5 to f_{d_5} remains active.*

integers, denote the depth from the input and output layers, l being the depth of the latent space layer. In our network architecture, encoder and decoder layers with $i = j$ have to match, i.e., the output shape of f_{e_i} has to be identical to that of f_{d_j} and vice versa. This setup allows for a greedy, layer-wise pretraining of the autoencoder, as proposed by Bengio et al. [BLPL07], where beginning from a shallow single layer deep autoencoder, additional layers are added to the model forming a series of deeper models for each stage. The optimization problem of such a stacked autoencoder in pretraining is therefore formulated as

$$\min_{\theta_{e_{0\dots k}},\theta_{d_{0\dots k}}} |f_{d_0} \circ f_{d_1} \circ \dots \circ f_{d_k}(f_{e_k} \circ f_{e_{k-1}} \circ \dots \circ f_{e_0}(\mathbf{x}(t))) - \mathbf{x}(t)|_2, \tag{5}$$

with $\theta_{e_{0\dots k}}, \theta_{d_{0\dots k}}$ denoting the parameters of the sub-stack for pretraining stage k, and $\circ$ denoting composition of functions. For our final models, we typically use a depth $l = 5$, and thus perform 6 runs of pretraining before training the complete model. This is illustrated in Fig. 2, where the paths from f_{e_k} over c_k to f_{d_k} are only active in pretraining stage k. After pretraining only the path f_{e_5} over c_5 to f_{d_5} remains active.

Layer	Kernel	Stride	Activation	Output	Features
Input				$\mathbf{r}/1$	d
f_{e_0}	4	2	Linear	$\mathbf{r}/2$	32
f_{e_1}	2	2	LeakyReLU	$\mathbf{r}/4$	64
f_{e_2}	2	2	LeakyReLU	$\mathbf{r}/8$	128
f_{e_3}	2	2	LeakyReLU	$\mathbf{r}/16$	256
f_{e_4}	2	2	LeakyReLU	$\mathbf{r}/32$	512
f_{e_5}	2	2	LeakyReLU	$\mathbf{r}/64$	1024
c_5				$\mathbf{r}/64$	1024
f_{d_5}	2	2	LeakyReLU	$\mathbf{r}/32$	512
f_{d_4}	2	2	LeakyReLU	$\mathbf{r}/16$	256
f_{d_3}	2	2	LeakyReLU	$\mathbf{r}/8$	128
f_{d_2}	2	2	LeakyReLU	$\mathbf{r}/4$	64
f_{d_1}	2	2	LeakyReLU	$\mathbf{r}/2$	32
f_{d_0}	4	2	Linear	$\mathbf{r}/1$	d

Table 1: *Parameters of the autoencoder layers. Here, $\mathbf{r} \in \mathbb{N}^3$ denotes the resolution of the data, and $d \in \mathbb{N}$ its dimensionality.*

In addition, our autoencoder does not use any pooling layers, but instead only relies on strided convolutional layers. This means we apply convolutions with a stride of s, skipping $s-1$ entries when applying the convolutional kernel. We assume the input is padded, and hence for $s = 1$ the output size matches the input, while choosing a stride $s > 1$ results in a downsampled output [ODO16]. Equivalently the decoder network employs strided transposed convolutions, where strided application increases the output dimensions by a factor of s. The details of the network architecture, with corresponding strides and kernel sizes can be found in Table 1.

In addition to this basic architecture, we will also evaluate a *variational* autoencoder [RMW14] in Sec. 5 that enforces a normalization on the latent space while keeping the presented AE layout identical. As no pre-trained models for physics problems are available, we found greedy pre-training of the autoencoder stack to be crucial for a stable and feasible training process.

3.2. Prediction of Future States

The prediction network, that models $\tilde{f}_t$, transforms a sequence of $n+1$ chronological, encoded input states $X = (\mathbf{c}^{t-nh}, \dots, \mathbf{c}^{t-h}, \mathbf{c}^t)$ into a consecutive list of o predicted future states $Y = (\mathbf{c}^{t+h}, \dots, \mathbf{c}^{t+oh})$. The minimization problem solved during training thus aims for minimizing the mean absolute error between the o predicted and ground truth states with an L_1 norm:

$$\min_{\theta_t} \left| \tilde{f}_t\left(\mathbf{c}^{t-nh}, \dots, \mathbf{c}^{t-h}, \mathbf{c}^t\right) - \left[\mathbf{c}^{t+h}, \dots, \mathbf{c}^{t+oh}\right] \right|_1 . \tag{6}$$

Here θ_t denotes the parameters of the prediction network, and $[\cdot,\cdot]$ denotes concatenation of the $\mathbf{c}$ vectors.

In contrast to the spatial reduction network above, which receives the full spatial input at once and infers a latent space coordinate without any data internal to the network, the prediction network uses a recurrent architecture for predicting the evolution over time. It receives a series of inputs one by one, and computes its output iteratively with the help of an internal network state. In contrast to the spatial reduction, which very heavily relies on convolutions, we cannot employ similar convolutions for the time data sets. While it is a valid assumption that each entry of a latent space vector $\mathbf{c}$ varies smoothly in time, the order of the entries is arbitrary and we cannot make any assumptions about local neighborhoods within $\mathbf{c}$. As such, convolving $\mathbf{c}$ with a filter along the latent space entries typically does not give meaningful results. Instead, our prediction network will use convolutions to translate the LSTM state into the latent space, in addition to fully connected layers of LSTM units.

The prediction network approximates the desired function $\tilde{f}_t$ with the help of an internal temporal context of dimension m_t, which we will denote as $\mathbf{d}$. Thus, the first part of our prediction network represents a recurrent encoder module, transforming $n+1$ latent space points into a temporal context $\mathbf{d}$. The time decoder module has a similar structure, and is likewise realized with layers of LSTM units. This module takes a context $\mathbf{d}$ as input, and outputs a series of future, spatial latent space representations. By means of its internal state, the time decoder is trained to predict o future states when receiving the same context $\mathbf{d}$ repeatedly. We use *tanh* activations for all LSTM layers, and hard sigmoid functions for efficient, internal activations.

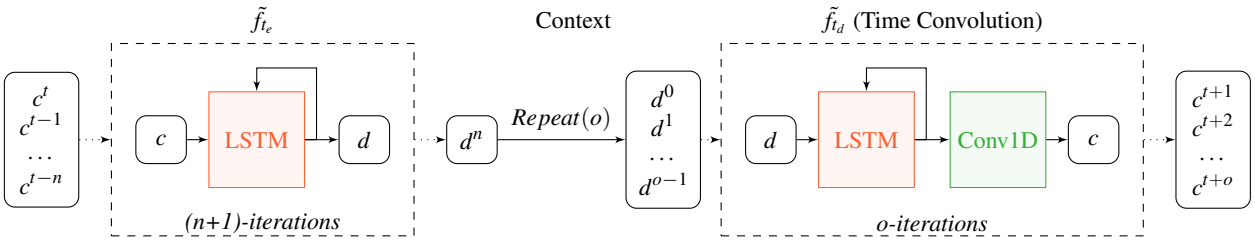


Figure 3: *Architecture overview of our LSTM prediction network. The dashed boxes indicate an iterative evaluation. Layer details can be found in the supplemental document.*

Note that the iterative nature is shared by encoder and decoder module of the prediction network, i.e., the encoder actually internally produces $n+1$ contexts, the first n of which are intermediate contexts. These intermediate contexts are only required for the feedback loop internal to the corresponding LSTM layer, and are discarded afterwards. We only keep the very last context in order to pass it to the decoder part of the network. This context is repeated o times, in order for the decoder LSTM to infer the desired future states.

Despite the spatial dimensionality reduction with an autoencoder, the number of weights in LSTM layers can quickly grow due to their inherent internal feedback loops (typically equivalent to four fully connected layers). To prevent overfitting from exploding weight numbers in the LSTM layers, we propose a hybrid structure of LSTM units and convolutions as shown in Fig. 3 that is used instead of the fully recurrent approach presented in Fig. 4.

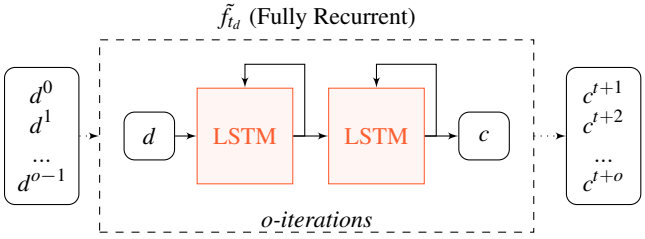


Figure 4: *Alternative to the time convolution sequence to sequence decoder (fully recurrent)*

For our prediction network we use two LSTM layers that infer an internal temporal representation of the data, followed by a final linear, convolutional network that translates the temporal representation into the corresponding latent space point. This convolution effectively represents a translation of the context information from the LSTM layer into latent space points that is constant for all output steps. This architecture effectively prevents overfitting, and ensures a high quality temporal prediction, as we will demonstrate below. In particular, we will show that this hybrid network outperforms networks purely based on LSTM layers, and significantly reduces the weight footprint. Additionally the prediction network architecture can be extended by applying multiple stacked convolution layers after the final LSTM layer. We found this hybrid architecture crucial for inferring the high-dimensional outputs of physical simulations.

While the autoencoder, thanks to its fully convolutional architecture, could be applied to inputs of varying size, the prediction network is trained for fixed latent space inputs, and internal context sizes. Correspondingly, when the latent space size m_s changes, it influences the size of the prediction network's layers. Hence, the prediction network has to be re-trained from scratch when the latent space size is changed. We have not found this critical in practice, because the prediction network takes significantly less time to train than the autoencoder, as we will discuss in Sec. 5.

4. Fluid Flow Data

To generate fluid data sets for training we rely on a NS solver with operator splitting [Bri15] to calculate the training data at discrete points in space and time. On a high level, the solver contains the following steps: computing motion of the fluid with the help of transport, i.e. *advection* steps, for the velocity $\mathbf{u}$, evaluating external forces, and then computing the harmonic pressure function p. In addition, a visible, passive quantity such as smoke density ρ, or a level-set representation ϕ for free surface flows is often advected in parallel to the velocity itself. Calculating the pressure typically involves solving an elliptic second-order PDE, and the gradient of the resulting pressure is used to make the flow divergence free.

In addition, we consider a *split* pressure, which represents a residual quantity. Assuming a fluid at rest on a ground with height z_g, a hydrostatic pressure p_s for cell at height z, can be calculated as $p_s(z) = p(z_0) + \frac{1}{A}\int_z^{z_0}\iint_A \mathbf{g}\rho(h)\,\mathrm{d}x\mathrm{d}y\,\mathrm{d}h$, with z_0, p_0, A denoting surface height, surface pressure, and cell area, respectively. As density and gravity can be treated as constant in our setting, this further simplifies to $p_s = \rho\mathbf{g}(z - z_0)$. While this form can be evaluated very efficiently, it has the drawback that it is only valid for fluids in hydrostatic equilibrium, and typically cannot be used for dynamic simulations in 3D. Given a data-driven method to predict pressure fields, we can incorporate the hydrostatic pressure into a 3D liquid simulation by decomposing the regular pressure field into hydrostatic and dynamic components $p_t = p_s + p_d$, such that our autoencoder separately receives and encodes the two fields p_s and p_d. To differentiate between the classic pressure field created by the simulation and our extracted split pressure components p_s and p_d, the classic pressure field is called total pressure p_t in the following. With this split pressure, the autoencoder could potentially put more emphasis on the small scale fluctuations p_d from the hydrostatic pressure gradient.

Overall, these physical data sets differ significantly from data

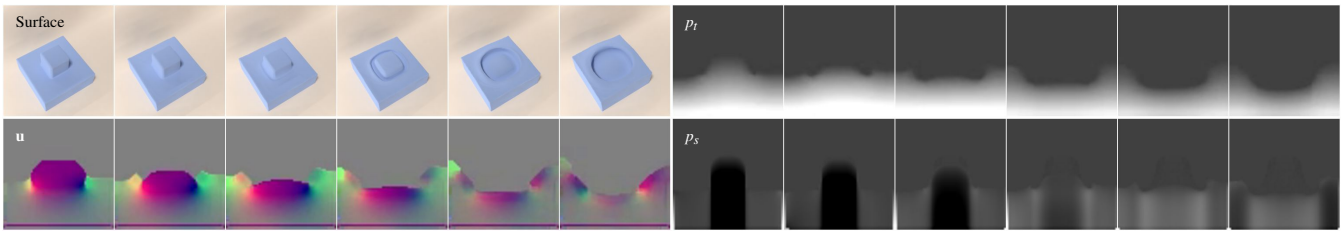


Figure 5: *Example sequences of different quantities under consideration, all ground truth reference data (center slices). The surface shown top left illustrates the 3D configuration of the liquid, but is not used for inference. The three velocity components of* **u** *are shown as RGB, whereas the total pressure p_t and the static part of the split pressure p_s are shown as grayscale images.*

sets such as natural images that are targeted with other learning approaches. They are typically well structured, and less ambiguous due to a lack of projections, which motivates our goal to use learned models. At the same time they exhibit strong temporal changes, as is visible in Fig. 5, which make the temporal inference problem a non-trivial task.

Depending on the choice of physical quantity to infer with our framework, different simulation algorithms emerge. We will focus on velocity **u** and the two pressure variants, total p_t and split (p_s and p_d) in the following. When targeting **u** with our method, this means that we can omit velocity advection as well as pressure solve, while the inference of pressure means that we only omit the pressure solve, but still need to perform advection and velocity correction with the pressure gradient. While this pressure inference requires more computations, the pressure solve is typically the most time consuming part with a super-linear complexity, and as such both options have comparable runtimes. When predicting the pressure field, we also use a boundary condition alignment step for the free surface [ATW15]. It takes the form of three Jacobi iterations in a narrow band at the liquid surface in order to align the Dirichlet boundary conditions with the current position of the interface. This step is important for liquids, as it incorporates small scale dynamics, leaving the large-scale dynamics to a learned model.

4.1. Interval Prediction

A variant for both of these simulation algorithm classes is to only rely on the network prediction for a limited time interval of i_p time steps, and then perform a single full simulation step without any network calculations, i.e., for $i_p = 0$ the network is not used at all, while $i_p = \infty$ is identical to the full network prediction described in the previous paragraph. We will investigate prediction intervals on the order of 4 to 14 steps. This simulation variant represents a joint numerical time integration and network prediction, that can have advantages to prevent drift from the learned predictions. We will denote such versions as *interval predictions* below.

4.2. Data Sets

To demonstrate that our approach is applicable to a wide range of physics phenomena, we will show results with three different 3D data sets in the following. To ensure a sufficient amount of variance with respect to physical motions and dynamics, we use randomized simulation setups. We target scenes with high complexity, i.e., strong visible splashes and vortices, and large CFL (Courant-Friedrichs-Lewy) numbers (typically around 2-3), that measure how fast information travels from cell to cell in a complex simulation domain. For each of our data sets, we generate n_s scenes of different initial conditions, for which we discard the first n_w time steps, as these typically contain small and regular, and hence less representative dynamics. Afterwards, we store a fixed number of n_t time steps as training data, resulting in a final size of $n_s n_t$ spatial data sets. Each data set content is normalized to the range of [-1,1].

	liquid64	liquid128	smoke128
Scenes n_s	4000	800	800
Time steps n_t	100	100	100
Size	419.43GB	671.09GB	671.09GB
Size, encoded	1.64GB	2.62GB	2.62GB

Table 2: *List of the augmented data sets for the total pressure architecture. Compression by a factor of 256 is achieved by the encoder part of the autoencoder f_e.*

Two of the three data sets contain liquids, while the additional one targets smoke simulations. The liquid data sets with spatial resolutions of 64^3 and 128^3 contain randomized sloshing waves and colliding bodies of liquid. The scene setup consists of a low basin, represented by a large volume of liquid at the bottom of the domain, and a tall but narrow pillar of liquid, that drops into it. Additionally a random amount, ranging from zero to three smaller liquid drops are placed randomly in the domain. For the 128^3 data set we additionally include complex geometries for the initial liquid bodies, yielding a larger range of behavior. These data sets will be denoted as *liquid64* and *liquid128*, respectively. In addition, we consider a data set containing single-phase flows with buoyant smoke which we will denote as *smoke128*. We place 4 to 10 inflow regions into an empty domain at rest, and then simulate the resulting plumes of hot smoke. As all setups are invariant w.r.t. rotations around the axis of gravity (Y in our setups), we augment the data sets by mirroring along XY and YZ. This leads to sizes of the data sets from 80k to 400k entries, and the 128^3 data sets have a total size of 671GB. Rendered examples from all data sets can be found in Fig. 17, Fig. 18 and Fig. 19 in the supplemental document, as well as further information about the initial conditions and physical parameters of the fluids.

5. Evaluation and Training

In the following we will evaluate the different options discussed in the previous section with respect to their prediction accuracies. In terms of evaluation metrics, we will use PSNR (peak signal-to-noise ratio) as a baseline metric, in addition to a surface-based Hausdorff distance in order to more accurately compare the position of the liquid interface [HKR93, LDGN15]. More specifically, given two signed distance functions ϕ_r, ϕ_p representing reference and predicted surfaces, we compute the surface error as

$$e_h = \max(1/|S_p| \sum_{\mathbf{p}_1 \in S_p} \phi_r(\mathbf{p}_1), 1/|S_r| \sum_{\mathbf{p}_2 \in S_r} \phi_p(\mathbf{p}_2))/\Delta x. \quad (7)$$

Unless otherwise noted, the error measurements start after 50 steps of simulation, and are averaged for ten test scenes from the *liquid64* setup.

5.1. Spatial Encoding

We first evaluate the accuracy of only the spatial encoding, i.e., the autoencoder network in conjunction with a numerical time integration scheme. At the end of a fluid solving time step, we encode the physical variable $\mathbf{x}$ under consideration with $\mathbf{c} = f_e(\mathbf{x})$, and then restore it from its latent space representation $\mathbf{x}' = f_d(\mathbf{c})$. In the following, we will compare flow velocity $\mathbf{u}$, total pressure p_t, and split pressure (p_s, p_d), all with a latent space size of $m_s = 1024$. We train a new autoencoder for each quantity, and we additionally consider a variational autoencoder for the split pressure. Training times for the autoencoders were two days on average, including pre-training. To train the different autoencoders, we use 6 epochs of pretraining and 25 epochs of training using an Adam optimizer, with a learning rate of 0.001 and a decay factor of 0.005. For training we used 80% of the data set, 10% for validation during training, and another 10% for testing.

Fig. 7a and Fig. 7e show error measurements averaged for 10 simulations from the test data set. Given the complexity of the data, especially the total pressure variant exhibits very good representational capabilities with an average PSNR value of 69.14. On the other hand, the velocity encoding introduces significantly larger errors in Fig. 7e. Interestingly, neither the latent space normalization of the VAE, nor the split pressure data increase the reconstruction accuracy, i.e., the CNN does not benefit from the reduced data range of the pressure splitting approach. A visual comparison of the results can be found in Fig. 6.

5.2. Temporal Prediction

Next, we evaluate reconstruction quality when including the temporal prediction network. Thus, now a quantity $\mathbf{x}'$ is inferred based on a series of previous latent space points. For the following tests, our prediction model uses a history of 6, and infers the next time step, thus $o = 1$, with a latent space size $m_s = 1024$. For a resolution of 64^3 the fully recurrent network contains 700, and 1500 units for the first and second LSTM layer of Fig. 3, respectively. Hence, $\mathbf{d}$ has a dimensionality of 700 for this setup. The two LSTM layers are followed by a convolutional layer targeting the m_s latent space dimensions for our hybrid architecture, or alternatively another LSTM layer of size m_s for the fully recurrent version. A dropout rate of $1.32 \cdot 10^{-2}$ with a recurrent dropout of 0.385, and a learning rate of $1.26 \cdot 10^{-4}$ with a decay factor of $3.34 \cdot 10^{-4}$ were used for all trainings of the prediction network. Training was run for 50 epochs with RMSProp, with 319600 training samples in each epoch, taking 2 hours, on average. Hyperparameters as well as the length of the time history used as input for the prediction network and the generated output time steps were chosen by utilizing a hyper parameter search, i.e. training multiple configurations of the same network with differing input-/output counts or hyperparameter settings.

	Layer (Type)	Activation	Output Shape
$\tilde{f}_{t_e}$	Input		$(n+1, m_s)$
	LSTM	tanh	(m_t)
Context	Repeat		(o, m_t)
$\tilde{f}_{t_d}$	LSTM	tanh	(o, m_{t_d})
	Conv1D	linear	(o, m_s)

Table 3: *Recurrent prediction network with hybrid architecture (Fig. 3)*

	Layer (Type)	Activation	Output Shape
$\tilde{f}_{t_e}$	Input		$(n+1, m_s)$
	LSTM	tanh	(m_t)
Context	Repeat		(o, m_t)
$\tilde{f}_{t_d}$	LSTM	tanh	(o, m_{t_d})
	LSTM	tanh	(o, m_s)

Table 4: *Fully recurrent network architecture (Fig. 4)*

The error measurements for simulations predicted by the combination of autoencoder and prediction network are shown in Fig. 7b and Fig. 7f, with a surface visualization in Fig. 9. Given the autoencoder baseline, the prediction network does very well at predicting future states for the simulation variables. The accuracy only slightly decreases compared to Fig. 7a and 7e, with an average PSNR value of 64.80 (a decrease of only 6.2% w.r.t. the AE baseline). Here, it is also worth noting that the LSTM does not benefit from the normalized latent space of the VAE. On the contrary, the predictions without the regular AE exhibit a lower error.

Fig. 7c shows an evaluation of the interval prediction scheme explained above. Here we employ the LSTM for $i_p = 14$ consecutive steps, and then perform a single regular simulation step. This especially improves the pressure predictions, for which the average surface error after 100 steps is still below two cells.

We also evaluate how well our model can predict future states based on a single set of inputs. For multiple output steps, i.e. $o > 1$, our model predicts several latent space points from a single time context $\mathbf{d}$. A graph comparing accuracy for 1, 3 and 5 steps of output can be found in Fig. 7h. It is apparent that the accuracy barely degrades when multiple steps are predicted at once. However, this case is significantly more efficient for our model. E.g., the $o = 3$ prediction only requires 30% more time to evaluate, despite generating three times as many predictions (details can be found in Sec. 6). Thus, the LSTM context successfully captures the state of

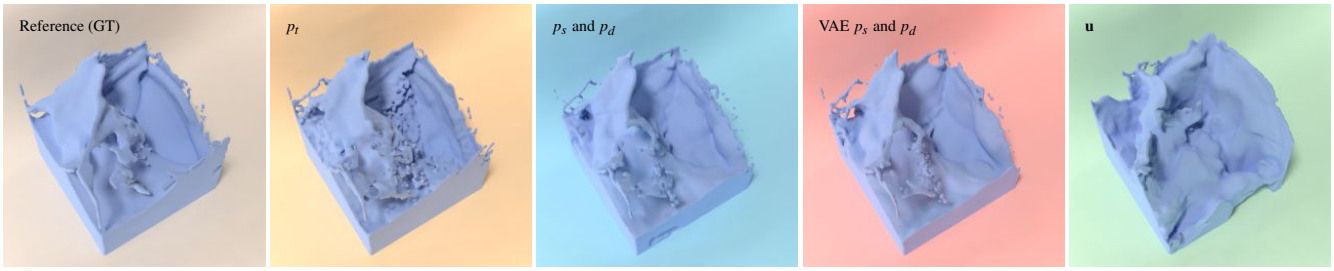
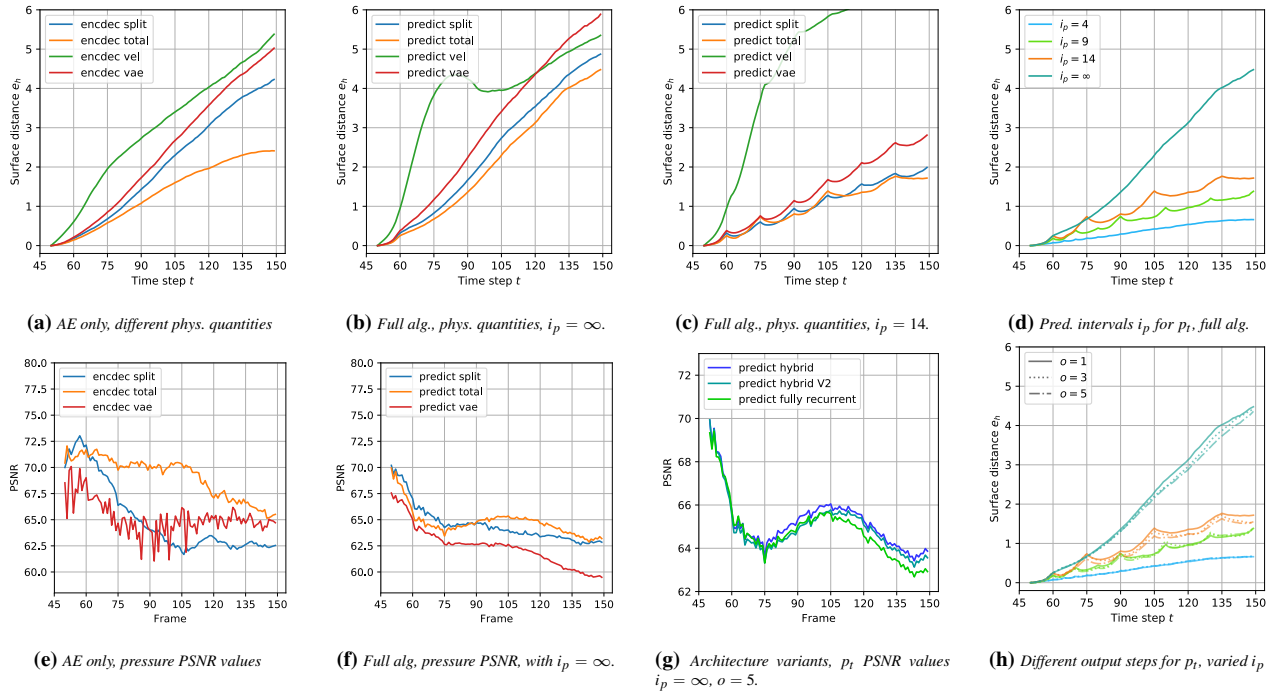


Figure 6: *Comparison of free surface renderings of simulations driven by a classic fluid simulation working with the different fields that are directly en- and then decoded, i.e. compressed, with a trained autoencoder. Time prediction is not used in this example, thus only the performance of the individual autoencoders for each quantity is evaluated. The velocity significantly differs from the reference (i.e., ground truth data, left), while all three pressure variants fare well with average PSNRs of 64.81, 64.55, and 62.45 (f.l.t.r.).*



(a) *AE only, different phys. quantities* **(b)** *Full alg., phys. quantities,* $i_p = \infty$. **(c)** *Full alg., phys. quantities,* $i_p = 14$. **(d)** *Pred. intervals* i_p *for* p_t*, full alg.*

(e) *AE only, pressure PSNR values* **(f)** *Full alg, pressure PSNR, with* $i_p = \infty$. **(g)** *Architecture variants,* p_t *PSNR values* $i_p = \infty$, $o = 5$. **(h)** *Different output steps for* p_t*, varied* i_p

Figure 7: *Error graphs over time for 100 steps, averaged over ten* liquid64 *simulations. Note that* $o = 1$ *in Fig. 7h corresponds to Fig. 7d, and is repeated for comparison.*

the temporal latent space evolution, such that the model can predict future states almost as far as the given input history.

A comparison of a fully recurrent LSTM with our proposed hybrid alternative can be found in Fig. 7g. In this scene, representative for our other test runs, the hybrid architecture outperforms the fully recurrent (FR) version in terms of accuracy, while using 8.9m fewer weights than the latter. The full network sizes are 19.5m weights for hybrid, and 28.4m for the FR network. We additionally evaluate a hybrid architecture with an additional conv. layer of size 4096 with tanh activation after the LSTM decoder layer (V2 in Fig. 7g). This variant yields similar error measurements to the original hybrid architecture. We found in general that additional layers did not significantly improve prediction quality in our setting. The FR version for the 128^3 data set below requires 369.4m weights due to its increased latent space dimensionality, which turned out to be infeasible. Our hybrid variant has 64m weights, which is still a significant number, but yields accurate predictions and reasonable training times. Thus, in the following tests, a total pressure inference model with a hybrid LSTM architecture for $o = 1$ will be used unless otherwise noted.

To clearly show the full data sets and their evolution over the course of a temporal prediction, we have trained a two-dimensional model, the details of which are given in App. B. In Fig. 8 sequences of the ground truth data are compared to the corresponding autoencoder baseline, and the outputs of our prediction network. Even though the autoencoder produces noise due to the strong compression, the temporal predictions closely match the autoencoder baseline, and the network is able to reproduce the complex behavior

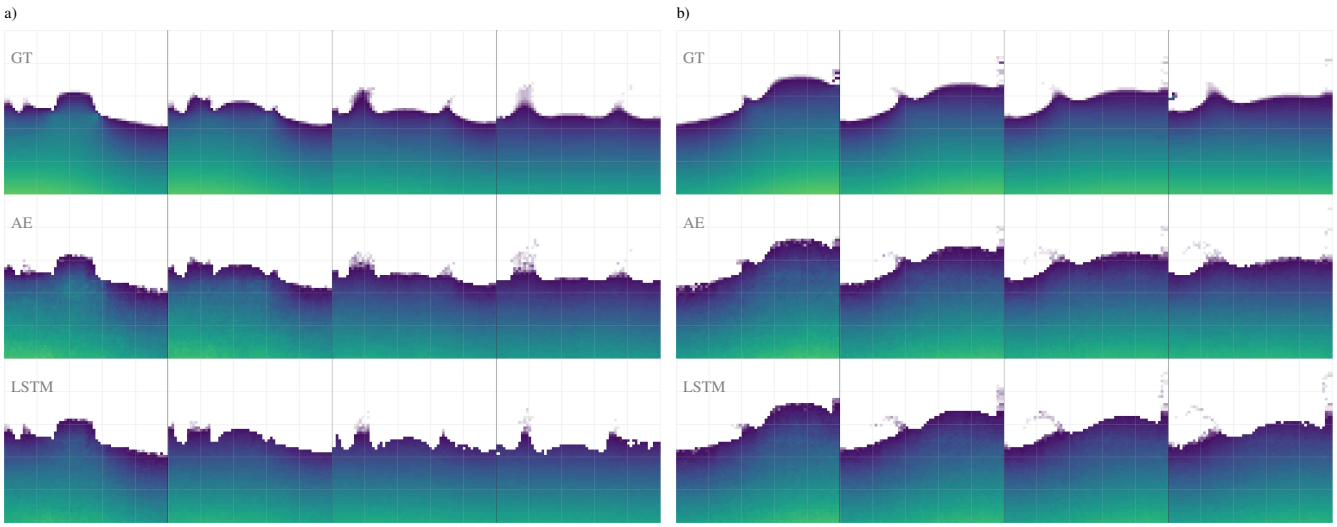


Figure 8: *Two examples of ground truth pressure fields (top), the autoencoder baseline (middle), and the LSTM predictions (bottom). Both examples have resolutions of* 64^2*, and are shown over the course of a long prediction horizon of 30 steps. The network successfully predicts the temporal evolution within the latent space with* $i_p = \infty$*, as shown in the bottom row.*

Figure 9: *Liquid surfaces predicted by different models for 40 steps with* $i_p = \infty$*. While the velocity version (green) leads to large errors in surface position, all three pressure versions closely capture the large scale motions. On smaller scales, both split pressure variants (*p_s *and* p_d*) introduce artifacts.*

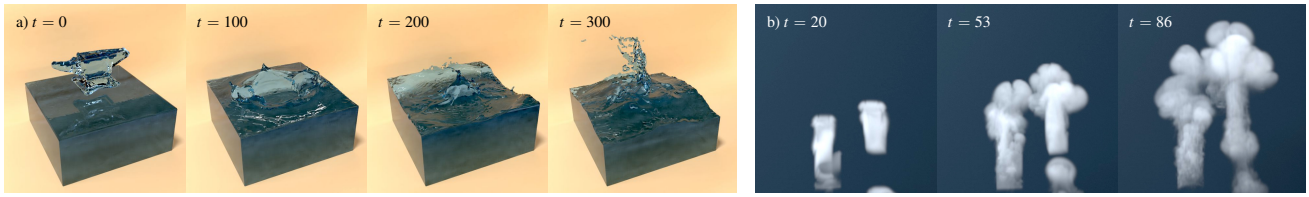


Figure 10: *a) A test simulation with our* liquid128 *model. The initial anvil shape was not part of the training data, but our model successfully generalizes to unseen shapes such as this one. b) A test simulation configuration for our* smoke128 *model.*

of the underlying simulations. E.g., the two waves forming on the right hand side of the domain in Fig. 8a indicate that the network successfully learned an abstraction of the temporal evolution of the flow. Further tests of the full prediction network with autoencoder models that utilize gradient losses to circumvent the visual noise, yielded no better prediction capabilities than the presented autoencoder with L2 loss. Additional 2D examples using the presented autoencoder can be found in Fig. 16 of App. B.

6. Results

We now apply our model to the additional data sets with higher spatial resolutions, and we will highlight the resulting performance in more detail. First, we demonstrate how our method performs on the *liquid128* data set, with its eight times larger number of degrees of freedom per volume. Correspondingly, we use a latent space size of $m_s = 8192$, and a prediction network with LSTM layers of size 1000 and 1500. Despite the additional complexity of this data set, our method successfully predicts the temporal evolution of the pressure fields, with an average PSNR of 44.8. The lower value compared to the 64^3 case is most likely caused by the higher intricacy of the 128^3 data. Fig. 10a) shows a more realistically rendered simulation for $i_p = 4$. This setup contains a shape that was not part of any training data simulations. Our model successfully handles this new configuration, as well as other situations shown in the accompanying video. This indicates that our model generalizes to a broad class of physical behavior. To evaluate long term stability, we have additionally simulated a scene for 650 time steps which successfully comes to rest. This simulation and additional scenes can be found in the supplemental video.

A trained model for the *smoke128* data set can be seen in Fig. 10b. Despite the significantly different physics, our approach successfully predicts the evolution and motion of the vortex structures. However, we noticed a tendency to underestimate pressure values, and to reduce small-scale motions. Thus, while our model successfully captures a significant part of the underlying physics, there is a clear room for improvement for this data set.

Our method also leads to significant speedups compared to regular pressure solvers, especially for larger volumes. For example the pressure inference by the prediction network for a 128^3 volume takes 9.5ms, on average. Including the times to encode and decode the respective simulation fields of resolution 128^3 (4.1ms and 3.3ms, respectively) this represents a 155× speedup compared to a parallelized state-of-the-art iterative MIC-CG pressure solver [Bri15], running with eight threads. While the latter yields a higher overall accuracy, and runs on a CPU instead of a GPU, it also represents a highly optimized numerical method. We believe the speedup of our LSTM version indicates a huge potential for very fast physics solvers with learned models.

It however, also leads to a degradation of accuracy compared to a regular iterative solver. The degradation can be controlled by chosing an appropiate prediction interval as described in Sec. 5.2 and can therefore be set according to the required accuracy. Even when taking into account a factor of ca. 10× for GPUs due to their better memory bandwidth, this leaves a speedup by a factor of more than 15×, pointing to a significant gain in efficiency for our LSTM-based prediction. In addition, we measured the speedup for our (not particularly optimized) implementation, where we include data transfer to and from the GPU for each simulation step. This very simple implementation already yields practical speedups of 10x for an interval prediction with $i_p = 14$. Details can be found in Table 5 and Table 6, while Table 2 summarizes the sizes of our data sets. All measurements were created with the *tensorflow timeline* tools on Intel i7 6700k (4GHz) and Nvidia Geforce GTX970.

Interval i_p	Solve	Mean surf. dist	Speedup
Reference	2.629s	0.0	1.0
4	0.600s	0.0187	4.4
9	0.335s	0.0300	7.8
14	0.244s	0.0365	10.1
∞	0.047s	0.0479	55.9
	Enc/Dec	Prediction	Speedup
Core exec. time	4.1ms + 3.3ms	9.5ms	155.6

Table 5: *Performance measurement of ten* liquid128 *example scenes, averaged over 150 simulation steps each. The mean surface distance is a measure of deviation from the reference per solve.*

	Solve		Speedup
Reference	169ms		1.0
	Enc/Dec	Prediction	
Core exec., $o = 1$	3.8ms + 3.2ms	9.6ms	10.2
Core exec., $o = 3$	3.9ms + 3 ∗ 3.3ms	12.5ms	19.3

Table 6: *Performance measurement of ten* liquid64 *example scenes, averaged over 150 simulation steps each.*

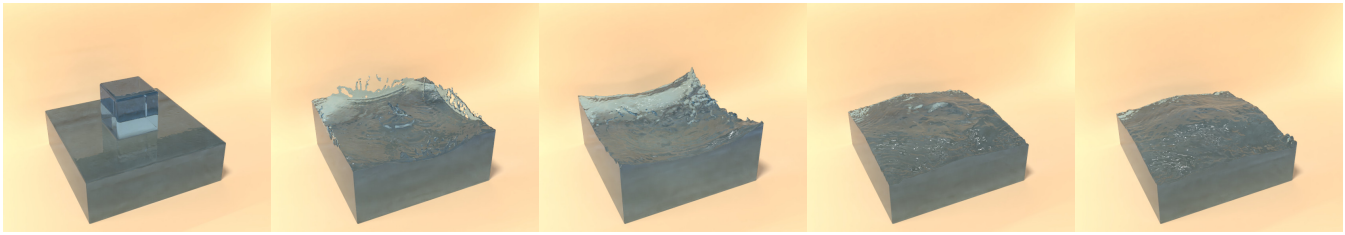
6.1. Limitations

While we have shown that our approach leads to large speed-ups and robust simulations for a significant variety of fluid scenes, there are several areas with room for improvements and follow up work. First, our LSTM at the moment strongly relies on the AE, which primarily encodes large scale scale dynamics, while small scale dynamics are integrated by the alignment of free surface boundary conditions [ATW15]. Also, our current, relatively simple AE can introduce a certain amount of noise in the solutions, which, however, can potentially be alleviated by different network architectures.

Overall, improving the AE network is important in order to improve the quality of the temporal predictions. Our experiments also show that larger data sets should directly translate into improved predictions. This is especially important for the latent space data set, which cannot be easily augmented.

7. Conclusions

With this work we arrive at three important conclusions: first, deep neural network architectures can successfully predict the temporal



(a) $t = 0$ **(b)** $t = 200$ **(c)** $t = 400$ **(d)** $t = 600$ **(e)** $t = 800$

Figure 11: *Renderings of a long running prediction scene for the* liquid128 *data set with* $i_p = 4$. *The fluid successfully comes to rest at the end of the simulation after 800 time steps.*

evolution of dense physical functions, second, learned latent spaces in conjunction with LSTM-CNN hybrids are highly suitable for this task, and third, they can yield very significant increases in simulation performance.

In this way, we arrive at a data-driven solver that yields practical speed-ups, and at its core is more than 150x faster than a regular pressure solve. We believe that our work represents an important first step towards deep-learning powered simulation algorithms. On the other hand, given the complexity of the problem at hand, our approach represents only a first step. There are numerous, highly interesting avenues for future research, ranging from improving the accuracy of the predictions, over performance considerations, to using such physics predictions as priors for inverse problems.

References

[ATW15] ANDO R., THUEREY N., WOJTAN C.: A dimension-reduced pressure solver for liquid simulations. *Comp. Grap. Forum 34*, 2 (2015), 10. 3, 6, 10

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3 (July 2007). URL: http://doi.acm.org/10.1145/1276377.1276502, doi:10.1145/1276377.1276502. 3

[BLPL07] BENGIO Y., LAMBLIN P., POPOVICI D., LAROCHELLE H.: Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (2007), pp. 153–160. 4

[BPL*16] BATTAGLIA P., PASCANU R., LAI M., REZENDE D. J., ET AL.: Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems* (2016), pp. 4502–4510. 2

[Bri15] BRIDSON R.: *Fluid Simulation for Computer Graphics*. CRC Press, 2015. 3, 5, 10, 14

[CT17] CHU M., THUEREY N.: Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. Graph. 36(4)*, 69 (2017). 3

[CUTT16] CHANG M. B., ULLMAN T., TORRALBA A., TENENBAUM J. B.: A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341* (2016). 2

[ENGF03] ENRIGHT D., NGUYEN D., GIBOU F., FEDKIW R.: Using the Particle Level Set Method and a Second Order Accurate Pressure Boundary Condition for Free-Surface Flows. *Proc. of the 4th ASME-JSME Joint Fluids Engineering Conference* (2003). 3

[FAW*16] FERSTL F., ANDO R., WOJTAN C., WESTERMANN R., THUEREY N.: Narrow band flip for liquid simulations. In *Computer Graphics Forum* (2016), vol. 35(2), Wiley Online Library, pp. 225–232. 3

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proceedings of ACM SIGGRPAH* (2001), pp. 23–30. 3

[FGP17] FARIMANI A. B., GOMES J., PANDE V. S.: Deep learning the physics of transport phenomena. *arXiv:1709.02432* (2017). 2

[FM96] FOSTER N., METAXAS D.: Realistic Animation of Liquids. *Graphical Models and Image Processing 58*, 5 (Sept. 1996), 471–483. URL: http://dx.doi.org/10.1006/gmip.1996.0039, doi:10.1006/gmip.1996.0039. 3

[FMB*17] FEI Y. R., MAIA H. T., BATTY C., ZHENG C., GRINSPUN E.: A multi-scale model for simulating liquid-hair interactions. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 56. 3

[FNPS16] FLYNN J., NEULANDER I., PHILBIN J., SNAVELY N.: Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 5515–5524. 2

[HKR93] HUTTENLOCHER D. P., KLANDERMAN G. A., RUCKLIDGE W. J.: Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence 15*, 9 (1993), 850–863. 7

[ICS*14] IHMSEN M., CORNELIS J., SOLENTHALER B., HORVATH C., TESCHNER M.: Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (2014), 426–435. 3

[KE18] KANI J. N., ELSHEIKH A. H.: Reduced order modeling of subsurface multiphase flow models using deep residual recurrent neural networks. *CoRR abs/1810.10422* (2018). URL: http://arxiv.org/abs/1810.10422, arXiv:1810.10422. 2

[KM90] KASS M., MILLER G.: Rapid, Stable Fluid Dynamics for Computer Graphics. *ACM Trans. Graph. 24*, 4 (1990), 49–55. 3

[KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012), NIPS, pp. 1097–1105. 2

[KTJG08] KIM T., THUEREY N., JAMES D., GROSS M.: Wavelet Turbulence for Fluid Simulation. *ACM Trans. Graph. 27 (3)* (2008), 50:1–6. 3

[LDGN15] LI Y., DAI A., GUIBAS L., NIESSNER M.: Database-assisted object retrieval for real-time 3d reconstruction. In *Computer Graphics Forum* (2015), vol. 34(2), Wiley Online Library, pp. 435–446. 7

[LJS*15] LADICKY L., JEONG S., SOLENTHALER B., POLLEFEYS M., GROSS M.: Data-driven fluid simulations using regression forests. *ACM Trans. Graph. 34*, 6 (2015), 199. 2, 3

[LKB18] LUSCH B., KUTZ J. N., BRUNTON S. L.: Deep learning for universal linear embeddings of nonlinear dynamics. In *Nature Communications* (2018). 2

[LKT16] LING J., KURZAWSKI A., TEMPLETON J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics 807* (10 2016). doi:10.1017/jfm.2016.615. 2

[LLMD17] LONG Z., LU Y., MA X., DONG B.: Pde-net: Learning pdes from data. *arXiv:1710.09668* (2017). 2

[LPSB17] LUAN F., PARIS S., SHECHTMAN E., BALA K.: Deep photo style transfer. *arXiv preprint arXiv:1703.07511* (2017). 2

[LZF10] LENTINE M., ZHENG W., FEDKIW R.: A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM Trans. Graph.* (2010), vol. 29(4), ACM, p. 114. 3

[MBM*16] MNIH V., BADIA A. P., MIRZA M., GRAVES A., LILLICRAP T., HARLEY T., SILVER D., KAVUKCUOGLU K.: Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning* (2016), pp. 1928–1937. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based Fluid Simulation for Interactive Applications. In *Symposium on Computer Animation* (2003), pp. 154–159. 3

[MHN13] MAAS A. L., HANNUN A. Y., NG A. Y.: Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (2013). 3

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Trans. Graph. 33*, 4 (2014), 153. 3

[MMCS11] MASCI J., MEIER U., CIREŞAN D., SCHMIDHUBER J.: Stacked convolutional auto-encoders for hierarchical feature extraction. *Proc. ICANN* (2011), 52–59. 3

[MST10] MCADAMS A., SIFAKIS E., TERAN J.: A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *Symposium on Computer Animation* (2010), SCA '10, pp. 65–74. 3

[MTP*18] MA P., TIAN Y., PAN Z., REN B., MANOCHA D.: Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 96. 2, 3

[MWJK18] MORTON J., WITHERDEN F. D., JAMESON A., KOCHENDERFER M. J.: Deep dynamical modeling and control of unsteady fluid flows. *arXiv preprint arXiv:1805.07472* (2018). 2

[ODO16] ODENA A., DUMOULIN V., OLAH C.: Deconvolution and checkerboard artifacts. *Distill* (2016). URL: http://distill.pub/2016/deconv-checkerboard, doi:10.23915/distill.00003. 4

[RHW88] RUMELHART D. E., HINTON G. E., WILLIAMS R. J.: Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1. 2

[RMC16] RADFORD A., METZ L., CHINTALA S.: Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR* (2016). 2

[RMW14] REZENDE D. J., MOHAMED S., WIERSTRA D.: Stochastic backpropagation and approximate inference in deep generative models. In *Proc. ICML* (2014), pp. II–1278–II–1286. URL: http://dl.acm.org/citation.cfm?id=3044805.3045035. 4

[SCW*15] SHI X., CHEN Z., WANG H., YEUNG D., WONG W., WOO W.: Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR abs/1506.04214* (2015). URL: http://arxiv.org/abs/1506.04214, arXiv:1506.04214. 2

[SF17] SCHENCK C., FOX D.: Reasoning about liquids via closed-loop simulation. *arXiv:1703.01656* (2017). 2

[Sta99] STAM J.: Stable Fluids. In *Proc. ACM SIGGRAPH* (1999), ACM, pp. 121–128. 3

[SVL14] SUTSKEVER I., VINYALS O., LE Q. V.: Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2014), NIPS'14, MIT Press, pp. 3104–3112. 2

[SWH*16] SAITO S., WEI L., HU L., NAGANO K., LI H.: Photorealistic facial texture inference using deep neural networks. *arXiv preprint arXiv:1612.00523* (2016). 2

[TLK16] TENG Y., LEVIN D. I., KIM T.: Eulerian solid-fluid coupling. *ACM Trans. Graph. 35*, 6 (2016), 200. 3

[TLP06] TREUILLE A., LEWIS A., POPOVIĆ Z.: Model reduction for real-time fluids. *ACM Trans. Graph. 25*, 3 (July 2006), 826–834. 3

[TSSP16] TOMPSON J., SCHLACHTER K., SPRECHMANN P., PERLIN K.: Accelerating eulerian fluid simulation with convolutional networks. *arXiv: 1607.03597* (2016). 2, 3

[UGB*16] UPCHURCH P., GARDNER J., BALA K., PLESS R., SNAVELY N., WEINBERGER K.: Deep feature interpolation for image content changes. *arXiv preprint arXiv:1611.05507* (2016). 2

[UHT17] UM K., HU X., THUEREY N.: Splash modeling with neural networks. *arXiv:1704.04456* (2017). 3

[WZW*17] WATTERS N., ZORAN D., WEBER T., BATTAGLIA P., PASCANU R., TACCHETTI A.: Visual interaction networks. In *Advances in Neural Information Processing Systems* (2017), pp. 4540–4548. 2

[WZX*16] WU J., ZHANG C., XUE T., FREEMAN B., TENENBAUM J.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (2016), pp. 82–90. 2

[XFCT18] XIE Y., FRANZ E., CHU M., THUEREY N.: tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *SIGGRAPH* (2018). 2

[XYZM17] XU J., YAO T., ZHANG Y., MEI T.: Learning multimodal attention lstm networks for video captioning. In *Proceedings of the 2017 ACM on Multimedia Conference* (2017), ACM, pp. 537–545. 2

[YYX16] YANG C., YANG X., XIAO X.: Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds 27*, 3-4 (2016), 415–424. 3

[ZB05] ZHU Y., BRIDSON R.: Animating Sand as a Fluid. *ACM Trans. Graph. 24*, 3 (2005), 965–972. 3, 14

Supplemental Document for Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow

Appendix A: Long-short Term Memory Units and Dimensionality

A central challenge for deep learning problems involving fluid flow is the large number of degrees of freedom present in three-dimensional data sets. This quickly leads to layers with large numbers of nodes – from hundreds to thousands per layer. Here, a potentially unexpected side effect of using LSTM nodes is the number of weights they require.

Therefore we briefly summarize the central equations for layers of LSTM units to gain understanding about the required weights. Below, f,i,o,g,s will denote forget, input, output, update and result connections, respectively. h denotes the result of the LSTM layer. Subscripts denote time steps, while θ and b denote weight and bias. Below, we assume tanh as output activation function. The new state for time step t of an LSTM layer is then given by:

$$\begin{aligned} f_t &= \sigma(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f) \\ i_t &= \sigma(\theta_{xi} x_t + \theta_{hi} h_{t-1} + b_i) \\ o_t &= \sigma(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o) \\ g_t &= \tanh(\theta_{xg} x_t + \theta_{hg} h_{t-1} + b_g) \\ s_t &= f_t \odot s_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(s_t) \end{aligned} \tag{8}$$

The local feedback loops for the gates of an LSTM unit all have trainable weights, and as such induce an $n \times n$ weight matrix for n LSTM units. E.g., even for a simple network with a one dimensional input and output, and a single hidden layer of 1000 LSTM units, with only 2×1000 connections and weights between in-, output and middle layer, the LSTM layer internally stores 1000^2 weights for its temporal feedback loop. In practice, LSTM units have *input, forget and output* gates in addition to the feedback connections, leading to $4n^2$ internal weights for an LSTM layer of size n. Correspondingly, the number of weights of such a layer with n_o nodes, i.e., outputs, and n_i inputs is given by $n_{\text{lstm}} = 4(n_o^2 + n_o(n_i+1))$. In contrast, the number of weights for the 1D convolutions we propose in the main document is $n_{\text{conv-1d}} = n_o k(n_i+1)$, with a kernel size $k = 1$.

Keeping the number of weights at a minimum is in general extremely important to prevent overfitting, reduce execution times, and to arrive at networks which are able to generalize. To prevent the number of weights from exploding due to large LSTM layers, we propose the mixed use of LSTM units and convolutions for our final temporal network architecture. Here, we change the decoder part of the network to consist of a single dense LSTM layer that generates a sequence of o vectors of size m_{t_d}. Instead of processing these vectors with another dense LSTM layer as before, we concatenate the outputs into a single tensor, and employ a single one-dimensional convolution translating the intermediate vector dimension into the required m_s dimension for the latent space. Thus, the 1D convolution works along the vector content, and is applied in the same way to all o outputs. Unlike the dense LSTM layers, the 1D convolution does not have a quadratic weight footprint, and purely depends on the size of input and output vectors.

Appendix B: Additional Results

In Fig. 12 additional time-steps of the comparison from Fig. 9 are shown. Here, different inferred simulation quantities can be compared over the course of a simulation for different models. In addition, Fig. 13, 14, and 15 show more realistic renderings of our *liquid64*, *liquid128*, and *smoke128* models, respectively.

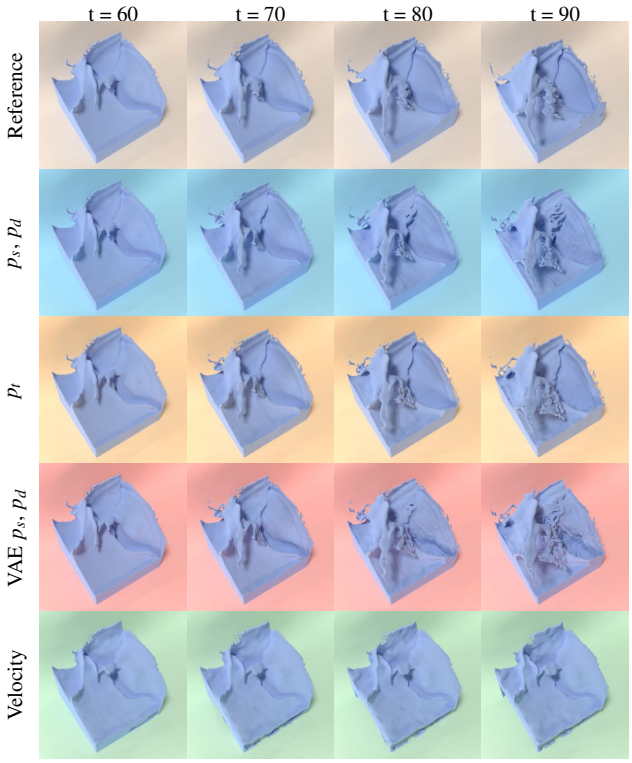
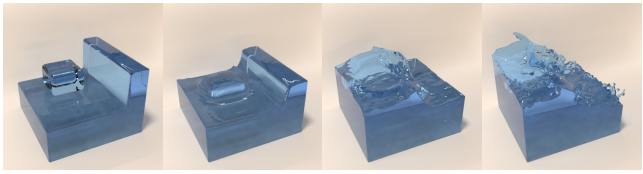


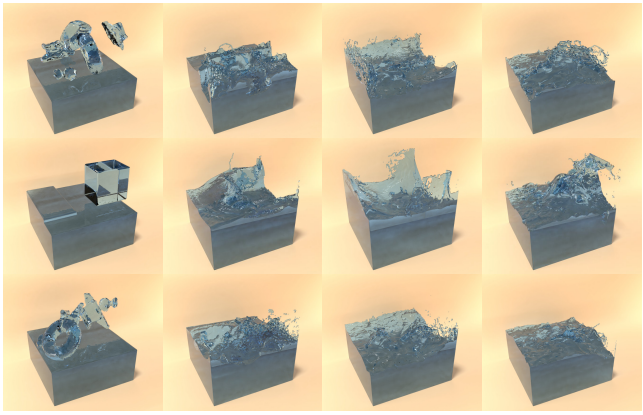
Figure 12: *Additional comparison of liquid surfaces predicted by different architectures for 40 time steps for $i_p = \infty$, with prediction starting at time step 50. While the velocity version (green) leads to large errors in surface position, all three pressure versions closely capture the large scale motions. On smaller scales, both split pressure and especially VAE introduce artifacts.*

As our solve indirectly targets divergence, we also measured how well the predicted pressure fields enforce divergence freeness over time. As a baseline, the numerical solver led to a residual divergence of $3.1 \cdot 10^{-3}$ on average. In contrast, the pressure field predicted by our LSTM on average introduced a $2.1 \cdot 10^{-4}$ increase



$t = 0$ $t = 50$ $t = 100$ $t = 150$

Figure 13: *Renderings at different points in time of a* 64^3 *scene predicted with* $i_p = 4$ *by our network.*



$t = 0$ $t = 100$ $t = 200$ $t = 300$

Figure 14: *Additional examples of* 128^3 *liquid scenes predicted with an interval of* $i_p = 4$ *by our LSTM network.*

of divergence per time step. Thus, the per time step error is well below the accuracy of our reference solver, and especially in combination with the interval predictions, we did not notice any significant changes in mass conservation compared to the reference simulations.

To visualize the temporal prediction capabilities as depicted in Fig. 8, the spatial encoding task of the total pressure p_t approach was reduced to two spatial dimensions. For this purpose a 2D autoencoder network was trained on a dataset of resolution 64^2. The temporal prediction network was trained as described in the main document. Additional sequences of the ground truth, the autoencoder baseline, and the temporal prediction by the LSTM network are shown in Fig. 16.

Appendix C: Fluid Simulation Setup

FLIP Simulation

In addition to Sec. 4, we provide more information on the simulation setup in the following. To generate our liquid datasets we use a classic NS solver [Bri15]. The timestep is fixed to 0.1, and pressure is computed with a conjugent gradient solver accuracy of $5 \cdot 10^{-5}$. The external forces in our setup only consist of a gravity vector of $(0.0, -0.01, 0.0)$ that is applied after every velocity advection step. No additional viscosity or surface tension forces are included.

$t = 0$ $t = 33$ $t = 66$ $t = 99$

Figure 15: *Several examples of* 128^3 *smoke scenes predicted with an interval of* $i_p = 3$ *by our LSTM network.*

In addition to the central quantities of a fluid solve, flow velocity $\mathbf{u}$, pressure p, and potentially visible quantities such as the levelset ϕ, we utilize the Fluid Implicit Particle (FLIP) [ZB05] method, which represents a grid-particle hybrid. It is used in this work on the one hand to generate the liquid datasets and on the other to be the base of our neural network driven interval prediction simulation.

To give a general overview of how the simulation proceeds, we shortly describe the computations executed for every time step in the following. In each simulation step we first advect the FLIP particle system *PS*, the levelset ϕ and the velocity $\mathbf{u}$ itself with the current velocity grid. Afterwards a second levelset containing the particle surface is created based on the current *PS* configuration and is merged with ϕ. The merged levelset is extrapolated within a narrow band region of 3, as described in the main text. After the levelset transformations $\mathbf{u}$ is updated with the *PS* velocities and the external forces like gravity are applied on the result, followed by the enforcement of the static wall boundary conditions. Next, a pressure field p is computed via a Poisson solve using the divergence of $\mathbf{u}$ as right hand side. After completing the pressure solve, the gradient of the result ∇p is subtracted from $\mathbf{u}$ yielding an approximation of a divergence free version of $\mathbf{u}$. The *PS* velocities are updated based on the difference between post-advection version of $\mathbf{u}$ and the latest divergence free one.

Prediction Integration

The presented LSTM prediction framework supports predictions of different simulation fields from the FLIP simulation presented above. The supported fields are the final $\mathbf{u}$ at the end of the simulation loop, the solved pressure p and the decomposed version of p with p_s and p_d, i.e. the hydrostatic and dynamic components of the regular pressure field, respectively. For the prediction of these fields we supply multiple architectures that are compared in the main text. Those are the *total pressure*, *variational split pressure*, *split pressure* and *velocity* versions. The difference between the variational split pressure and the default split pressure approach is the architec-

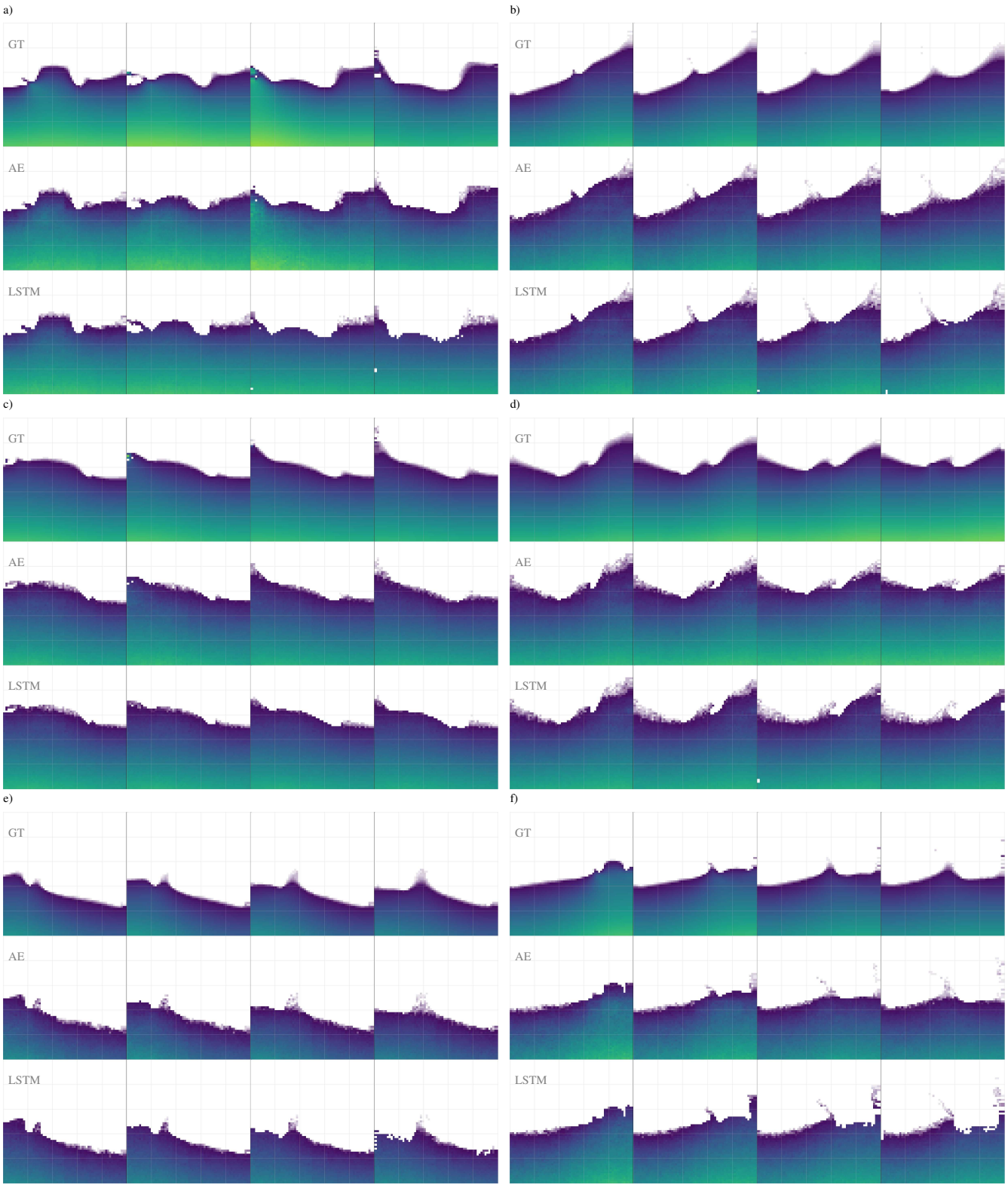


Figure 16: *Six additional example sequences of ground truth pressure fields (top), the autoencoder baseline (middle), and the LSTM predictions (bottom). All examples have resolutions of* 64^2*, and are shown over the course of a long horizon of 30 prediction steps with* $i_p = \infty$*. The LSTM closely predicts the temporal evolution within the latent space.*

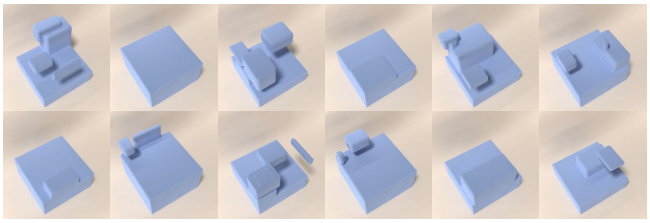


Figure 17: *Examples of initial scene states in the* liquid64 *data set.*

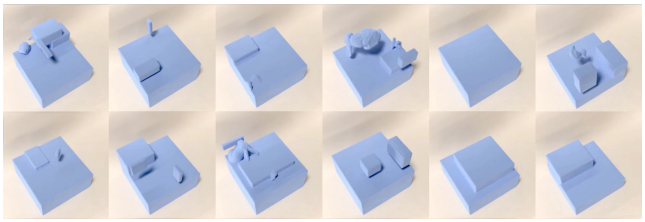


Figure 18: *Examples of initial scene states in the* liquid128 *data set. The more complex initial shapes are visible in several of these configurations.*

ture of the autoencoder, whereas the temporal prediction network stays the same.

Depending on the prediction architecture, the inferred, decoded predicted field is used instead of executing the corresponding numerical approximation step. When targeting **u** with our method, this means that we can omit velocity advection as well as pressure solve, while the inference of p by the split or total pressure architecture means that we only omit the pressure solve, but still need to perform advection and velocity correction with the pressure gradient. While the latter requires more computations, the pressure solve is typically the most time consuming part, with a super-linear complexity, and as such both options, using either **u** or the p variants, have comparable runtimes.

Appendix D: Hyperparameters

To find appropriate hyperparameters for the prediction network, a large number of training runs with varying parameters were executed on a subset of the total training data domain. The subset consisted of 100 scenes of the training data set discussed in Sec. 4.2.

In Fig. 20 (a-d), examples for those searches are shown. Each circle shown in the graphs represents the final result of one complete training run with the parameters given on the axes. The color

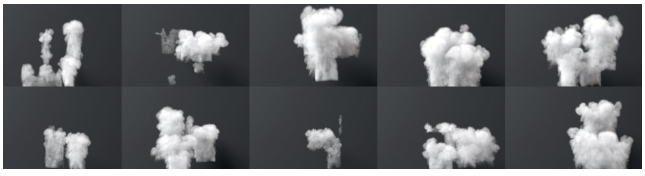
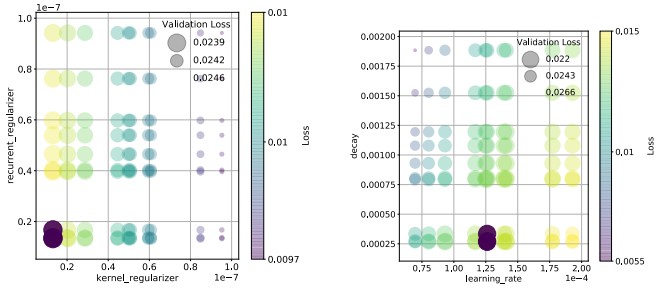
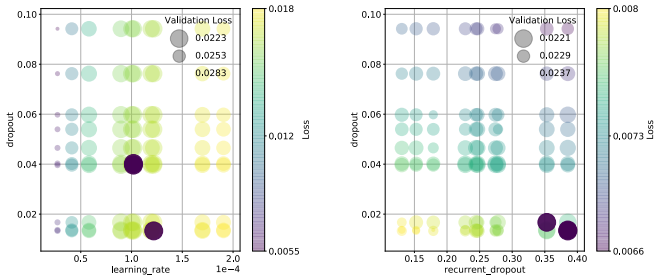


Figure 19: *Examples states of the* smoke128 *training data set at* $t = 65$.



(a) *kernel regularizer and recurrent regularizer* **(b)** *learning rate and learning rate decay*



(c) *learning rate and dropout* **(d)** *recurrent dropout and dropout*

Figure 20: *Random search of hyperparameters for the prediction network*

represents the mean absolute error of the training error, ranging from purple (the best) to yellow (the worst). The size of the circle corresponds to the validation error, i.e., the most important quantity we are interested in. The best two runs are highlighted with a dark coloring. These searches yield interesting results, e.g. Fig. 20a shows that the network performed best without any weight decay regularization applied.

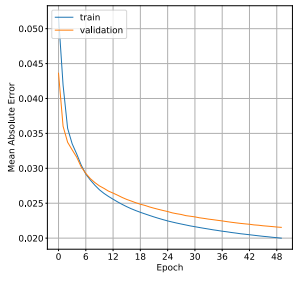


Figure 21: *Training history of the* liquid128 p_t *network.*

Choosing good parameters leads to robust learning behavior in the training process, an example is shown in Fig. 21. Note that it is possible for the validation error to be lower than the training error as dropout is not used for computing the validation loss. The mean absolute error of the prediction on a test set of 40 scenes, which was generated independently from the training and validation data, was 0.0201 for this case. These results suggest that the network generalizes well with the given hyperparameters.