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

Steffen Wiewel
Technical University of Munich
wiewel@in.tum.de

Moritz Becher
Technical University of Munich
mo.becher@tum.de

Nils Thuerey
Technical University of Munich
nils.thuerey@tum.de

Abstract

We propose a method for the data-driven inference of temporal evolutions of physical functions with deep learning. More specifically, we target fluid flow problems, and we propose a novel LSTM-based approach to predict the changes of the pressure field 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.

1 Introduction

The variables we use to describe real world physical systems often take the form of complex functions with high dimensionality, and we usually employ continuous models to describe how these functions evolve over time. In the following work we demonstrate that it is possible to instead use deep learning methods to infer physical functions. While such methods have been successful for sparse Lagrangian representations [2, 29, 6, 10], we will focus on high-dimensional Eulerian functions. More specifically, we will focus on the temporal evolution of physical functions that arise in the context of fluid flow. Fluids encompass a large and important class of materials in human environments, and as such they're particularly interesting candidates for learning models [26, 23].

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 reduced latent space [17, 21, 30]. We then train a modified sequence-to-sequence network [25, 18] that learns to predict future 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 non-linear representations. We will later on demonstrate that the setup for computing this reduced representation strongly influences how well the prediction network can infer changes over time, and we will demonstrate the generality of our approach with several liquid and single-phase problems. The specific contribu-

Preprint. This work is supported by the ERC Starting Grant 637014.

tions of this work are: a hybrid LSTM-CNN architecture to predict large-scale 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 architectural choices and parameters.

2 Related Work and Background

While physical properties and interactions of objects have long been of interest for computer vision problems [24, 34, 13], addressing physics problems with deep learning algorithms is an area of particular interest in recent years. Several works have targeted predictions of Lagrangian objects, connecting them with image data. E.g., Battaglia et al. [2] introduced a network architecture to predict two-dimensional physics, that also can be employed for predicting object motions in videos [29]. Another line of work proposed a different architecture to encode and predict two-dimensional rigid bodies physics [6], and improved predictions for Lagrangian systems were targeted by Yu et al. [33]. Ehrhardt et al., on the other hand, use recurrent NNs to predict trajectories and their likelihood for balls in height-field environments [10]. While the methods above successfully predict various physical effects, they do not extend to the high-dimensional, Eulerian functions of materials such as fluids.

Fluids are ubiquitous in nature, and traditionally simulated by finding solutions to the *Navier-Stokes* (NS) model [11]. Its incompressible form is given by $\partial \mathbf{u}/\partial t + \mathbf{u} \cdot \nabla \mathbf{u} = -1/\rho \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g}$, $\nabla \cdot \mathbf{u} = 0$, 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.

The sequence-to-sequence methods which we 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 [25]. Recurrent neural networks are popular for control tasks [18], automatic video captioning [31], or generic sequence predictions [7]. While others have studied combinations of RNNs and CNNs [8, 20], we propose a hybrid architecture to decouple the prediction task from the latent space dimensions.

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. [14]. Predictions of liquid motions for robotic control, with a particular focus on pouring motions [23] were also targeted. Other graphics works have learned flow descriptors [9], or the statistics of splash formation [28]. While CNN-based methods for pressure projections were developed [26, 32], our contributions are largely orthogonal. Instead of targeting divergence freeness for a single instance in time, our work aims for the more general task of learning the temporal evolution of physical functions, which we demonstrate for pressure among other functions. An important difference is also that these methods, similar to other model reduction techniques [27], and POD-based methods [5] have so far has only been demonstrated for single-phase simulations. For all of these methods, handling the strongly varying boundary conditions of liquids remains an open challenge, and we demonstrate below that our approach works especially well for liquid dynamics.

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, a trained neural network model in our case, our goal is to predict the future state $\mathbf{x}(t + h)$ as closely as possible given a current state and a series of n previous states, i.e., $f_t\big(\mathbf{x}(t - nh), ..., \mathbf{x}(t - h), \mathbf{x}(t)\big) \approx \mathbf{x}(t + h)$. Due to the high dimensionality of $\mathbf{x}$, directly solving this formulation would be very costly. Thus, we employ

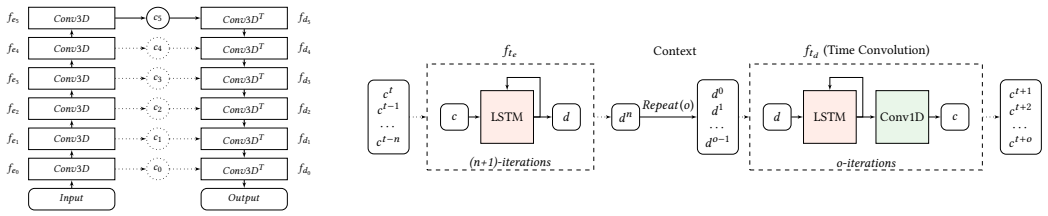


Figure 1: Architecture overview of our convolutional autoencoder (left), and LSTM prediction network (right). For the latter, the dashed boxes indicate an iterative evaluation. Layer details can be found in the supplemental document.

two additional functions f_d and f_e, that compute a low dimensional encoding. f_e maps into an m_s dimensional space $\mathbf{c} \in \mathbb{R}^{m_s}$ with $f_e(\mathbf{x}(t)) = \mathbf{c}$. Both functions are represented by trained neural networks, which are trained such that $f_d(f_e(\mathbf{x}(t))) \approx \mathbf{x}(t)$. 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 $f_d\Big(f_t\big(f_e(\mathbf{x}(t-nh)), ..., f_e(\mathbf{x}(t-h)), f_e(\mathbf{x}(t))\big)\Big) \approx \mathbf{x}(t+h)$. We will use CNNs for f_d and f_e, and thus $\mathbf{c}$ is given by their learned *latent space*. We choose its dimensionality m such that the temporal prediction problem above becomes feasible for dense three dimensional samples. Our prediction network will likewise employ an encoder-decoder structure, and turns the stack of encoded data sets $f_e(\mathbf{x})$ into a reduced representation $\mathbf{d}$, which we will refer to as time *context* below. An overview of our architecture can be found in Fig. 1.

3.1 Reducing Dimensionality

In order to reduce the spatial dimensionality of our inference problem, we employ a fully convolutional autoencoder architecture [17]. Our autoencoder (AE) consists of the aforementioned encoding and decoding functions f_e, f_d, in order to minimize $|f_d(f_e(\mathbf{x}(t))) - \mathbf{x}(t)|_2$. We use a series of convolutional layers activated by leaky rectified linear units (LeakyReLU) [16] 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 use the short form $\mathbf{c}^t = f_e(\mathbf{x}(t))$ to denote a spatial latent space point at time t.

In addition to this basic architecture, we will also evaluate a *variational* autoencoder below [22]. However, we found its latent space normalization to be counter productive for temporal prediction. 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

We propose to use a hybrid LSTM-CNN architecture for the temporal prediction task, which we found to yield better results than purely LSTM-based or convolutional approaches. This makes it possible to train networks that infer very high-dimensional outputs, and to the best of our knowledge, hybrid LSTM and convolutional networks have not been employed for latent-space prediction tasks up to now.

The prediction network 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:

$$\min_{\theta_t} \big| f_t\big(\mathbf{c}^{t-nh}, \dots, \mathbf{c}^{t-h}, \mathbf{c}^t\big) - \big[\mathbf{c}^{t+h}, \dots, \mathbf{c}^{t+oh}\big] \big|_1 \,. \tag{1}$$

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

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.

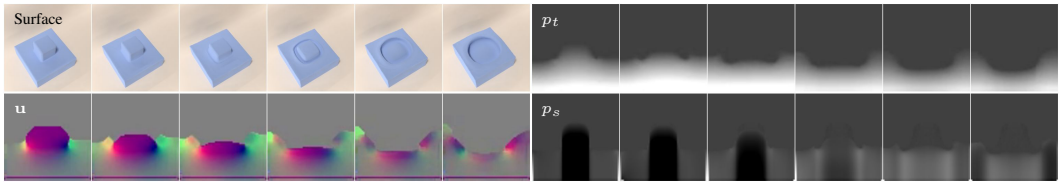


Figure 2: Example sequences of different quantities under consideration, all ground truth reference data (center slices). The surface shown top left illustrates the 3D configuration of the liquid, but is not used for inference. The three velocity components of $\mathbf{u}$ are shown as RGB.

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. For our prediction network we use two LSTM layers that infer an internal temporal representation of the data that is translated into the corresponding latent space point by a final linear, convolutional layer. 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. 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 [4] 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. A selection of input data sets over time is shown in Fig. 2. In addition, we consider a *split* pressure p_s, which represents a residual quantity. This pressure data has a smaller range of values, and contains fewer large scale gradients than the full pressure field. Our initial hypothesis was that this split pressure could reduce the learning and prediction difficulty, but our tests below indicate that the networks fare even better with the total pressure. Overall, these physical data sets differ significantly from data 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. 2, 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 $\mathbf{u}$ and pressure variants in the following. When targeting $\mathbf{u}$ 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

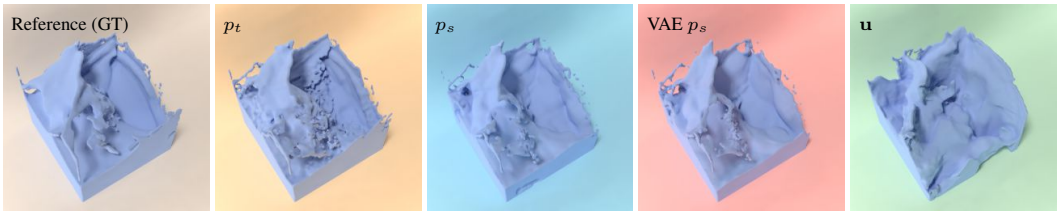


Figure 3: Different simulation fields en-/decoded with a trained AE, without the time prediction. The velocity significantly differs from the reference (i.e., ground truth data, left), while all three pressure variants fare well with average PSNRs of 64.81, 64.55, and 62.45 (f.l.t.r.).

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 [1]. 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.

A variant for both of these cases 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. 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.

4.1 Data Sets

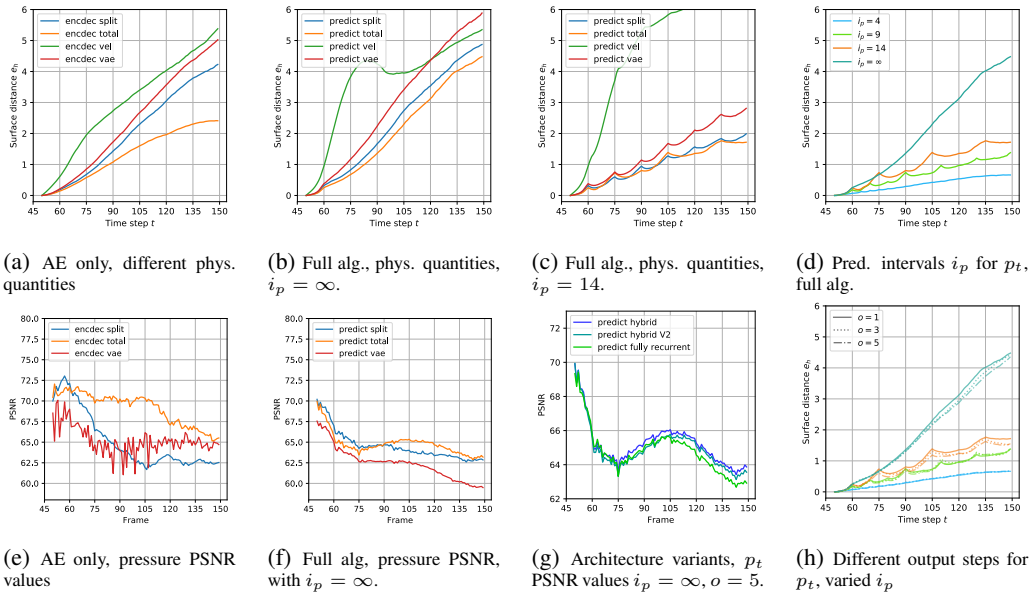
To demonstrate that our approach is applicable to a wide range of physics phenomena, we will show results with three different 3D data sets in the following. To ensure a sufficient amount of variance with respect to physical motions and dynamics, we use randomized simulation setups. We target scenes with high complexity, i.e., strong visible splashes and vortices, and large CFL numbers (typically around 2-3). 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].

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. 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. Examples from all data sets are given in the supplemental document.

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 as a baseline metric, in addition to a surface-based Hausdorff distance[1] in order to more accurately compare the position

[1] More specifically, given two signed distance functions ϕ_r, ϕ_p representing reference and predicted surfaces, we compute the surface error as $e_h = \max(1/|S_p| \sum_{\mathbf{p}_1 \in S_p} \phi_r(\mathbf{p}_1), 1/|S_r| \sum_{\mathbf{p}_2 \in S_r} \phi_p(\mathbf{p}_2))/\Delta x$.



(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 4: Error graphs over time for 100 steps, averaged over ten *liquid64* simulations. Note that $o = 1$ in Fig. 4h corresponds to Fig. 4d, and is repeated for comparison.

of the liquid interface [12, 15]. Unless otherwise noted, the error measurements start after 50 steps of simulation, and are averaged for ten test scenes from the *liquid64* setup.

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, 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 L_2 regularization with strength 0.005. For training we used 80% of the data set, 10% for validation during training, and another 10% for testing.

Fig. 4a and Fig. 4e 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. 4e. It is likewise obvious that neither the latent space normalization of the VAE, nor the split pressure data increase the reconstruction accuracy. A visual comparison of these results can be found in Fig. 3.

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. 1, 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 were selected with a broad search.

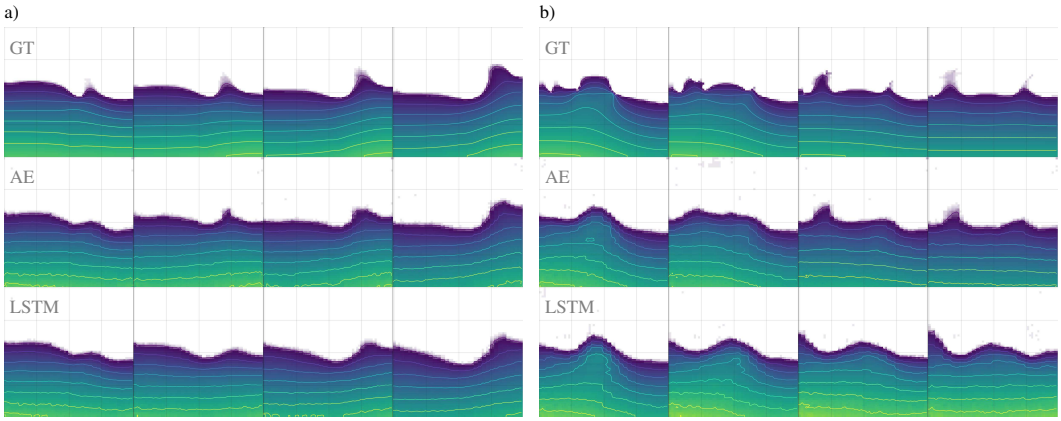


Figure 5: Two examples of ground truth pressure fields (top), the autoencoder baseline (middle), and the LSTM predictions (bottom). Both examples have resolutions of 64^2, and are shown over the course of a long prediction horizon of 30 steps. The network successfully predicts the temporal evolution within the latent space, as shown in the bottom row.

The error measurements for simulations predicted by the combination of autoencoder and prediction network are shown in Fig. 4b and Fig. 4f, with a surface visualization in Fig. 6. 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. 4a and 4e, with an average PSNR value of 64.80 (a decrease of only 6.2% w.r.t. the AE baseline). Fig. 4c 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. 4h. It is apparent that the accuracy barely degrades when multiple steps are predicted. 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 the supplemental document). Thus, the LSTM context successfully captures the state of 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. 4g. 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. 4g). 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, as we will demonstrate below. 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. D. In Fig. 5 sequences of the ground truth data are compared to the corresponding autoencoder baseline, and the outputs of our prediction network. The temporal predictions closely match the autoencoder baseline, and the network is able to reproduce the complex behavior of the underlying simulations. E.g., the wave forming on the right hand side of the domain in Fig. 5a indicates that the network successfully learned an abstraction of the temporal evolution of the flow. Additional 2D examples can be found in Fig. 10 of App. D.

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. 7a) 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.

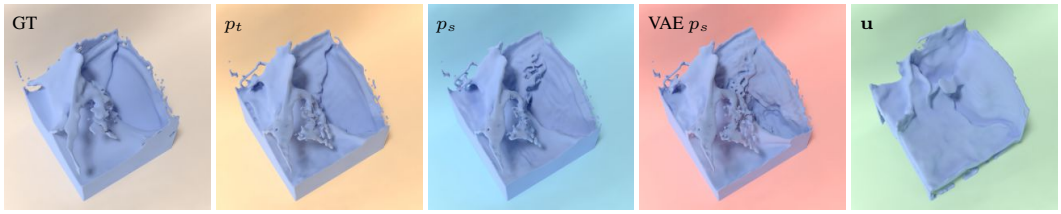


Figure 6: Liquid surfaces predicted by different models for 40 steps with $i_p = \infty$. While the velocity version (green) leads to large errors in surface position, all three pressure versions closely capture the large scale motions. On smaller scales, both p_s variants introduce artifacts.

A trained model for the *smoke128* data set can be seen in Fig. 7b). 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 speed-ups compared to regular pressure solvers, especially for larger volumes. Pressure inference for a 128^3 volume takes 9.5ms, on average, which represents a $155\times$ speed-up compared to a parallelized state-of-the-art iterative solver [4]. 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 speed up of our LSTM version indicates a huge potential for very fast physics solvers with learned models.

Discussion and Limitations While we have shown that our approach leads to large speed-ups and robust simulations for a significant variety of fluid scenes, it also has its set of limitations, and there are numerous interesting extensions for future 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 free surface treatment [1]. Thus, improving the AE network is important to improve the quality of the temporal predictions. Our experiments 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. Interestingly, our method arrives at a simulation algorithm that

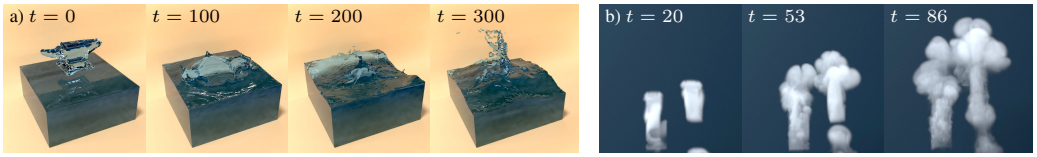


Figure 7: a) A test simulation with our *liquid128* model. The initial anvil shape was not part of the training data, but our model successfully generalizes to unseen shapes such as this one. b) A test simulation configuration for our *smoke128* model.

could not be formulated with traditional models: no formulation exists that allows for the calculation of a future pressure field only based on a history of previous fields.

7 Conclusion

With this work we arrive at three important conclusions: first, deep neural network architectures can successfully predict the temporal 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. We believe that our work represents an important first step towards deep-learning powered simulation algorithms. 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

[1] R. Ando, N. Thuerey, and C. Wojtan. A dimension-reduced pressure solver for liquid simulations. *Comp. Grap. Forum*, 34(2):10, 2015.

[2] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[3] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[4] R. Bridson. *Fluid Simulation for Computer Graphics*. CRC Press, 2015.

[5] T. Bui-Thanh, M. Damodaran, and K. E. Willcox. Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition. *AIAA journal*, 42(8):1505–1516, 2004.

[6] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341*, 2016.

[7] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

[8] J. Chen, L. Yang, Y. Zhang, M. Alber, and D. Z. Chen. Combining fully convolutional and recurrent neural networks for 3d biomedical image segmentation. In *Advances in Neural Information Processing Systems*, pages 3036–3044, 2016.

[9] M. Chu and N. Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. Graph.*, 36(4)(69), 2017.

[10] S. Ehrhardt, A. Monszpart, N. J. Mitra, and A. Vedaldi. Learning a physical long-term predictor. *arXiv:1703.00247*, 2017.

[11] J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.

[12] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863, 1993.

[13] Z. Jia, A. C. Gallagher, A. Saxena, and T. Chen. 3d reasoning from blocks to stability. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):905–918, 2015.

[14] L. Ladicky, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6):199, 2015.

[15] Y. Li, A. Dai, L. Guibas, and M. Nießner. Database-assisted object retrieval for real-time 3d reconstruction. In *Computer Graphics Forum*, volume 34(2), pages 435–446. Wiley Online Library, 2015.

[16] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[17] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59, 2011.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[19] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[20] D. Quang and X. Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, 44(11):e107–e107, 2016.

[21] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR*, 2016.

[22] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1278–II–1286. JMLR.org, 2014.

[23] C. Schenck and D. Fox. Reasoning about liquids via closed-loop simulation. *arXiv:1703.01656*, 2017.

[24] J. Schulman, A. Lee, J. Ho, and P. Abbeel. Tracking deformable objects with point clouds. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1130–1137. IEEE, 2013.

[25] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[26] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. *arXiv: 1607.03597*, 2016.

[27] A. Treuille, A. Lewis, and Z. Popović. Model reduction for real-time fluids. *ACM Trans. Graph.*, 25(3):826–834, July 2006.

[28] K. Um, X. Hu, and N. Thuerey. Splash modeling with neural networks. *arXiv:1704.04456*, 2017.

[29] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. Visual interaction networks. In *Advances in Neural Information Processing Systems*, pages 4540–4548, 2017.

[30] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[31] J. Xu, T. Yao, Y. Zhang, and T. Mei. Learning multimodal attention lstm networks for video captioning. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 537–545. ACM, 2017.

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

[33] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue. Long-term forecasting using tensor-train rnns. *arXiv preprint arXiv:1711.00073*, 2017.

[34] B. Zheng, Y. Zhao, C. Y. Joey, K. Ikeuchi, and S.-C. Zhu. Detecting potential falling objects by inferring human action and natural disturbance. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3417–3424. IEEE, 2014.

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

A Autoencoder Details

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 $i, j \in [0, n]$ denote the depth from the input and output layers, n 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. [3], 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

$$\min_{\theta_{e_{0\dots k}}, \theta_{d_{0\dots k}}} |f_{d_0} \circ f_{d_1} \circ \dots \circ f_{d_k}(f_{e_k} \circ f_{e_{k-1}} \circ \dots \circ f_{e_0}(\mathbf{x}(t))) - \mathbf{x}(t)|_2, \tag{2}$$

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 $n = 5$, and thus perform 6 runs of pretraining before training the complete model. This is illustrated in Fig. 1 of the main document, 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.

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 [19]. 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.

B Prediction Network Details

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

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.

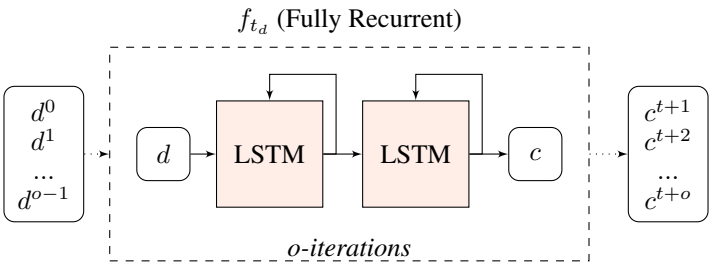


Figure 8: Alternative Sequence to Sequence Decoder (Fully Recurrent)

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 with the help of an internal temporal context with a 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 time 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. While the hybrid convolution version was already shown in the main document, the fully recurrent variant is illustrated in Fig. 8.

B.1 Dimensionality and Network Size

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. 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. 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.

LSTM units contain forget, input, output, update and result connections. Thus, the number of weights of an LSTM layer with n_o nodes, 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 $k = 1$. This architecture is summarized in Table 2.

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

Table 2: Recurrent prediction network with hybrid architecture (Main doc., Fig. 1)

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

Table 3: Fully recurrent network architecture (Fig. 8)

C Pressure Splitting

Although our final models directly infer the full pressure field, we have experimented with a pressure splitting approach, under the hypothesis that this could simplify the inference problem for the LSTM. This paragraphs explains details of our pressure splitting, as several of our tests in the main document use this data. 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 $p = p_s + p_d$, such that our autoencoder separately receives and encodes the two fields p_s and p_d. With this split pressure, the autoencoder could potentially put more emphasis on the small scale fluctuations p_d from the hydrostatic pressure gradient. The evaluations in the main document focus on p_s, where we have demonstrated that our approach does not benefit from this pressure splitting. We have also experimented with only inferring p_s, and using the analytically computed p_d fields, which, however, likewise did not lead to improved temporal inference.

D Additional Results

In Fig. 18 additional time-steps of the comparison from Fig. 6 are shown. Here, different inferred simulation quantities can be compared over the course of a simulation for different models. In addition, Fig. 15, 16, and 17 show more realistic renderings of our *liquid64*, *liquid128*, and *smoke128* models, respectively.

	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 4: 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.

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 diver-

gence 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 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. Below we additionally evaluate the accuracy of our method with respect to the pressure gradient, as it is crucial for removing divergent motions. Fig. 9 shows additional examples of the different physical quantities calculated with separately trained models, including boundary condition post processing.

Performance As outlined in the main document, our method leads to significant speed-ups compared to regular pressure solvers. Below, we will give additional details regarding these speed-ups. For the resolution of 128^3, the encoding and decoding stages take 4.1ms and 3.3ms [2], respectively. Evaluating the trained LSTM network itself is more expensive, but still very fast with 9.5ms, on average. In total, this execution is ca. $155\times$ faster than our CPU-based pressure solver[3].

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 + $3 * 3.3$ms	12.5ms	19.3

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

It however, also leads to a degradation of accuracy compared to a regular iterative solver. Even when taking into account a factor of ca. $10\times$ for GPUs due to their better memory bandwidth, this leaves a speedup by a factor of more than $15\times$, pointing to a significant gain in efficiency for our LSTM-based prediction. In addition, we measured the speed up 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 speed-ups of 10x for an interval prediction with $i_p = 14$. Details can be found in Table 5, while Table 4 summarizes the sizes of our data sets.

Temporal Prediction in 2D To visualize the temporal prediction capabilities as depicted in Fig. 5, 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. To increase the reconstruction quality of the input, a convolution layer with stride $(1, 1)$ was added before each of the $(2, 2)$ strided convolutions shown Fig. 1. Additionally, the loss was adjusted to put more emphasis on small values in the solution, so that the surface of the pressure field is more accurately represented. This is realized with a loss function $\mathrm{L}(x, y) = 0.9 \cdot \mathrm{MSE}(\tanh(x), \tanh(y)) + 0.1 \cdot \mathrm{MSE}(x, y)$. The temporal prediction network was trained as described above. Additional sequences of the ground

[2] Measured with the *tensorflow timeline* tools on Intel i7 6700k (4GHz) and Nvidia Geforce GTX970.

[3] We compare to a parallelized MIC-CG pressure solver [4], running with eight threads.

truth, the autoencoder baseline, and the temporal prediction by the LSTM network are shown in Fig. 10.

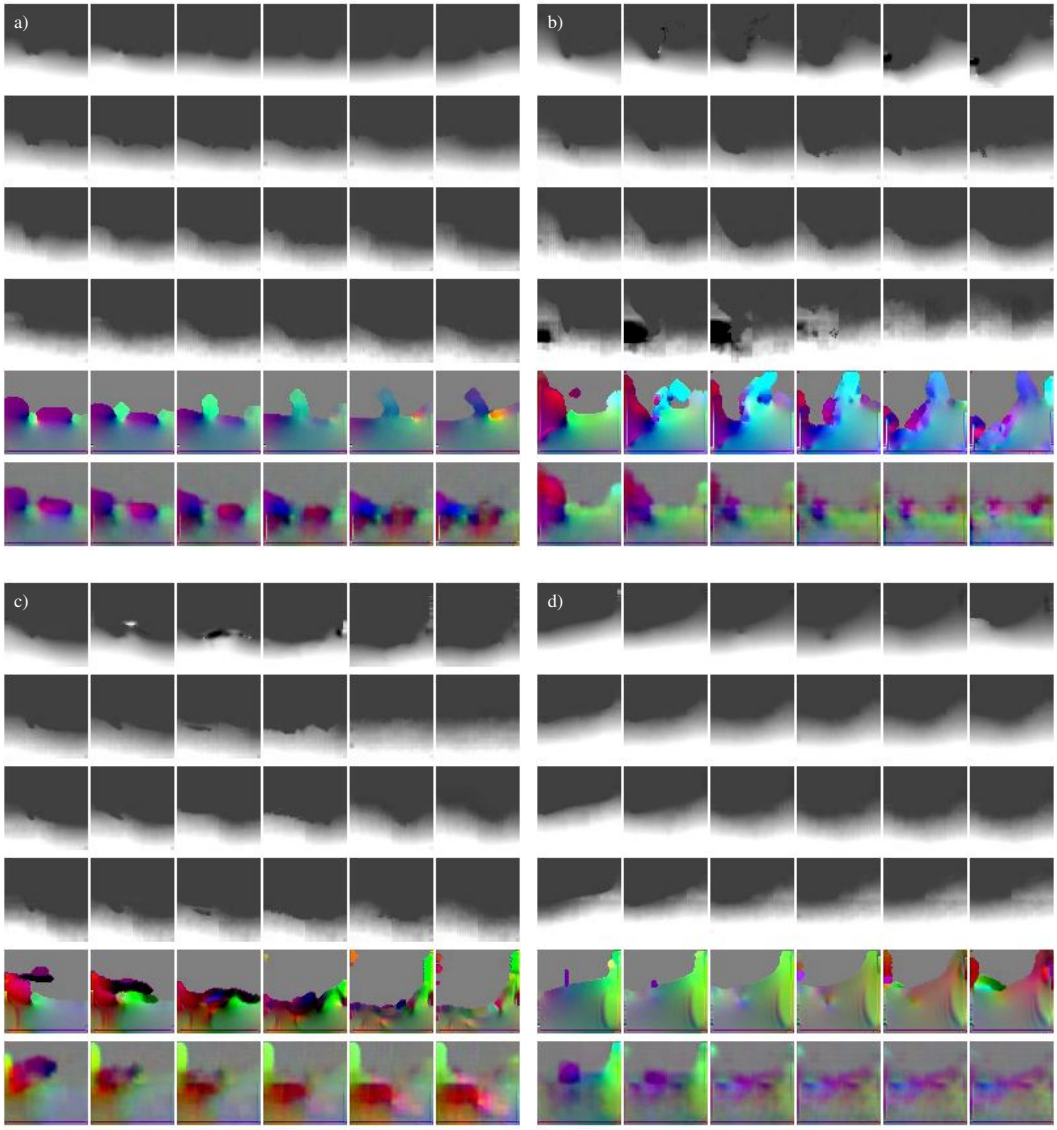


Figure 9: Four additional example sequences of different predicted quantities (a center slice through the 64^3 domain is shown). All were calculated with $i_p = \infty$, with $t = 1, 21, 42, 61, 81, 100$ for each column a block. The different states over time illustrate the complexity of the temporal changes of the target functions. The rows per block are from top to bottom: ground truth pressure, total pressure predicted, split pressure predicted, VAE split pressure predicted, ground truth velocity, and velocity predicted. Final pressure values are shown, i.e. after applying the boundary condition alignment. The larger errors for velocity and VAE split pressure are visible here. While all methods drift over time (comparing the right most column), the total pressure predictions (2nd from top) fare significantly better than their counterparts.

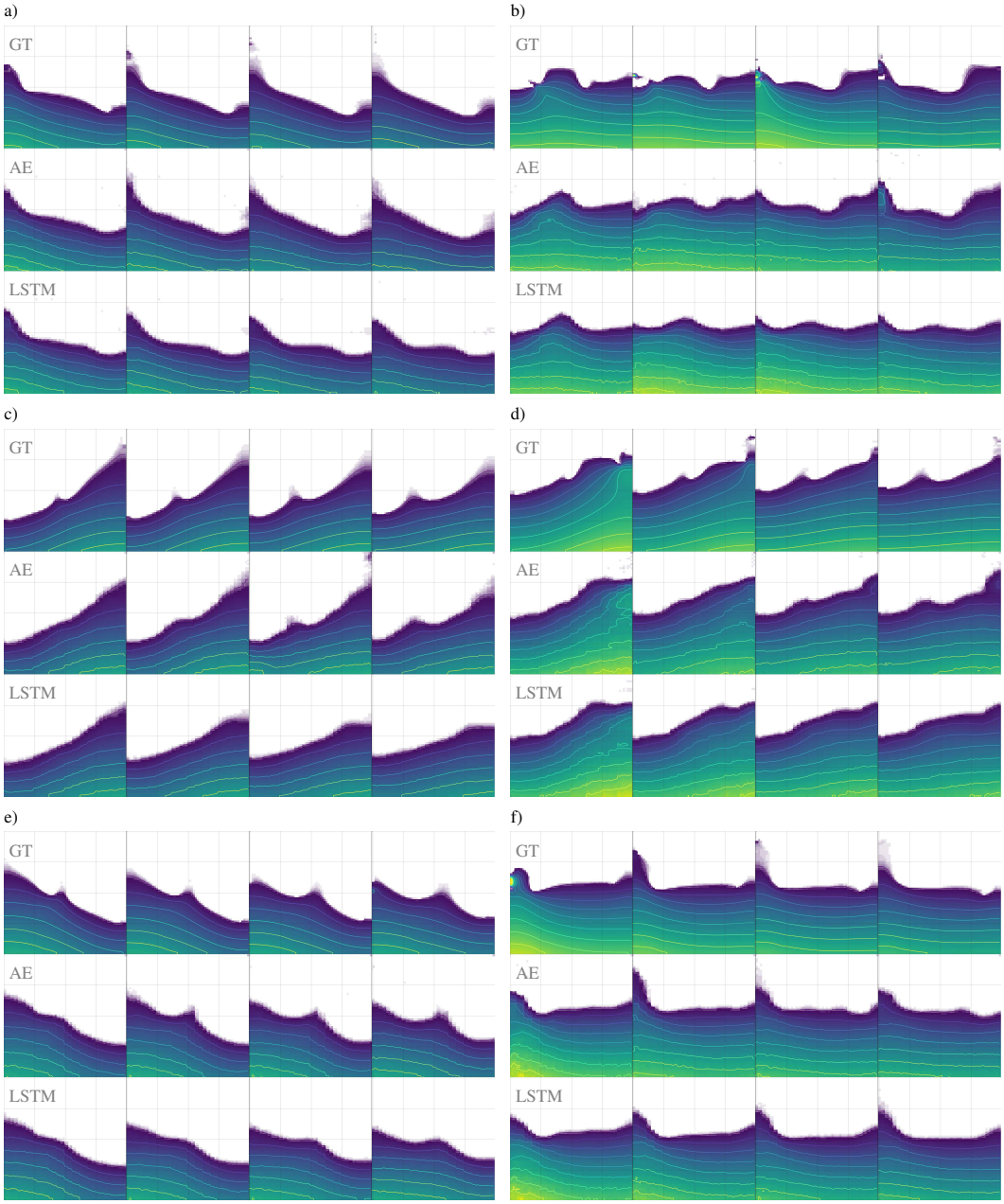


Figure 10: Six additional example sequences of ground truth pressure fields (top), the autoencoder baseline (middle), and the LSTM predictions (bottom). All examples have resolutions of 64^2, and are shown over the course of a long horizon of 30 prediction steps. The LSTM closely predicts the temporal evolution within the latent space.

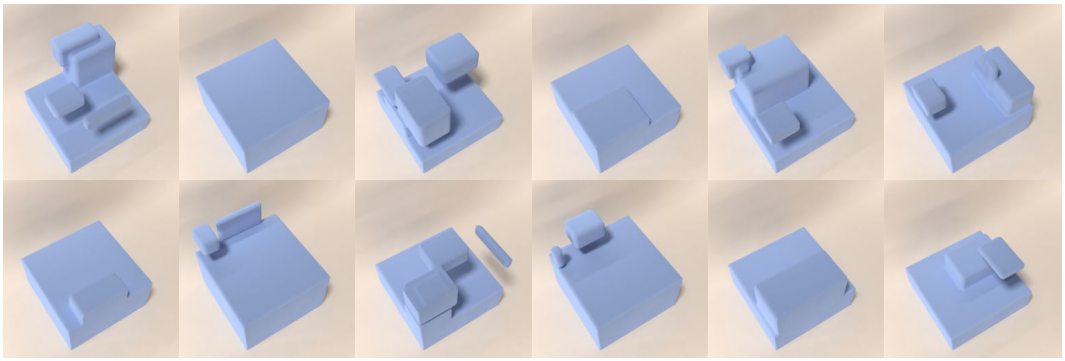


Figure 11: Examples of initial scene states in the *liquid64* data set.

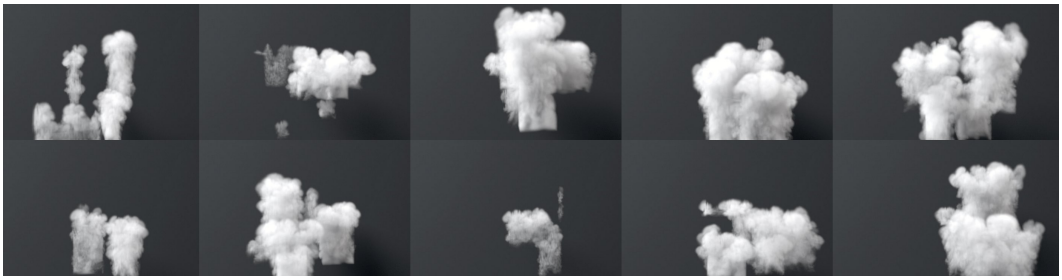


Figure 12: Examples states of the *smoke128* training data set at $t = 65$.

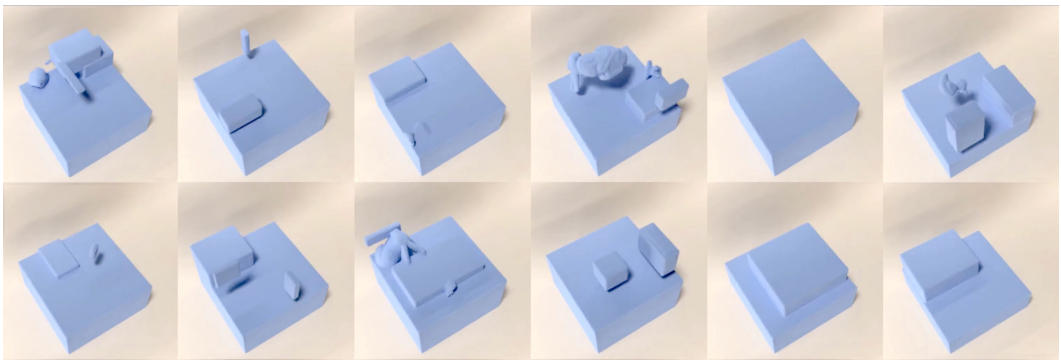
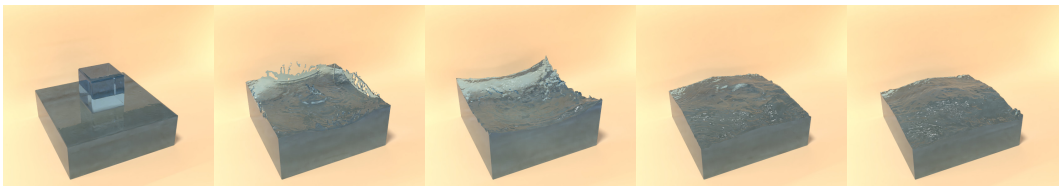
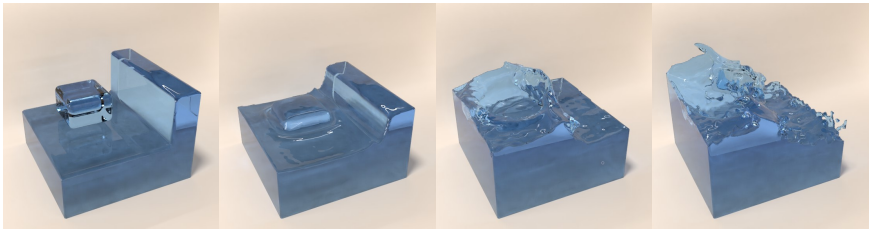


Figure 13: Examples of initial scene states in the *liquid128* data set. The more complex initial shapes are visible in several of these configurations.



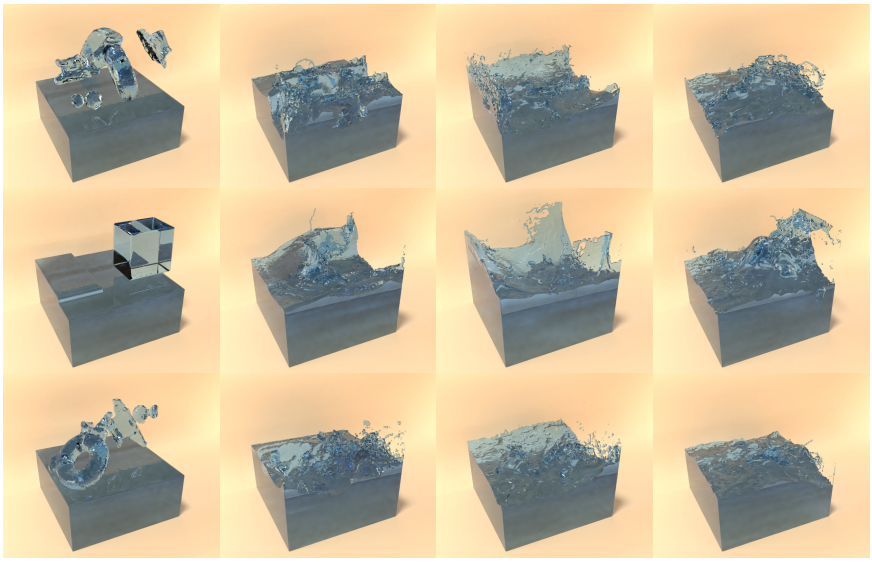
(a) $t = 0$ (b) $t = 200$ (c) $t = 400$ (d) $t = 600$ (e) $t = 800$

Figure 14: 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.



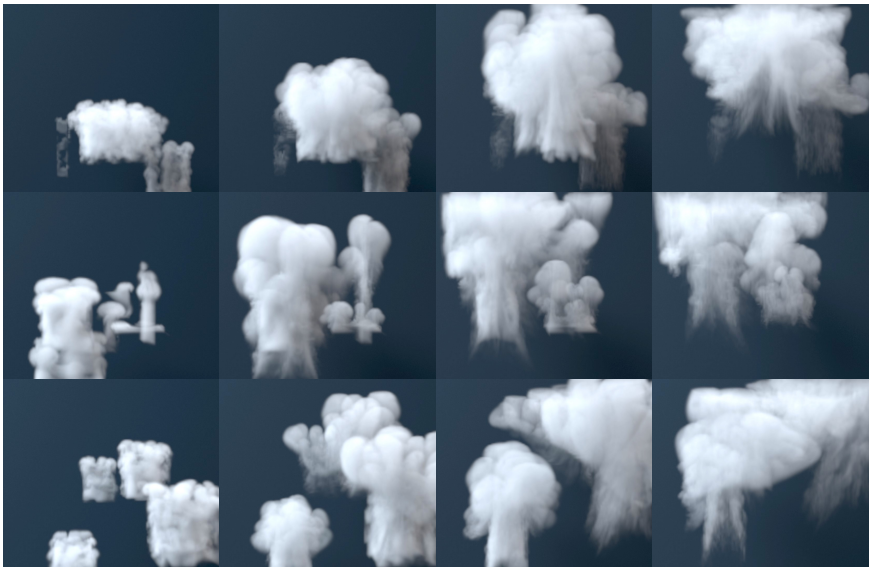
$t = 0$ $t = 50$ $t = 100$ $t = 150$

Figure 15: 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 16: Additional examples of 128^3 liquid scenes predicted with an interval of $i_p = 4$ by our LSTM network.



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

Figure 17: Several examples of 128^3 smoke scenes predicted with an interval of $i_p = 3$ by our LSTM network.

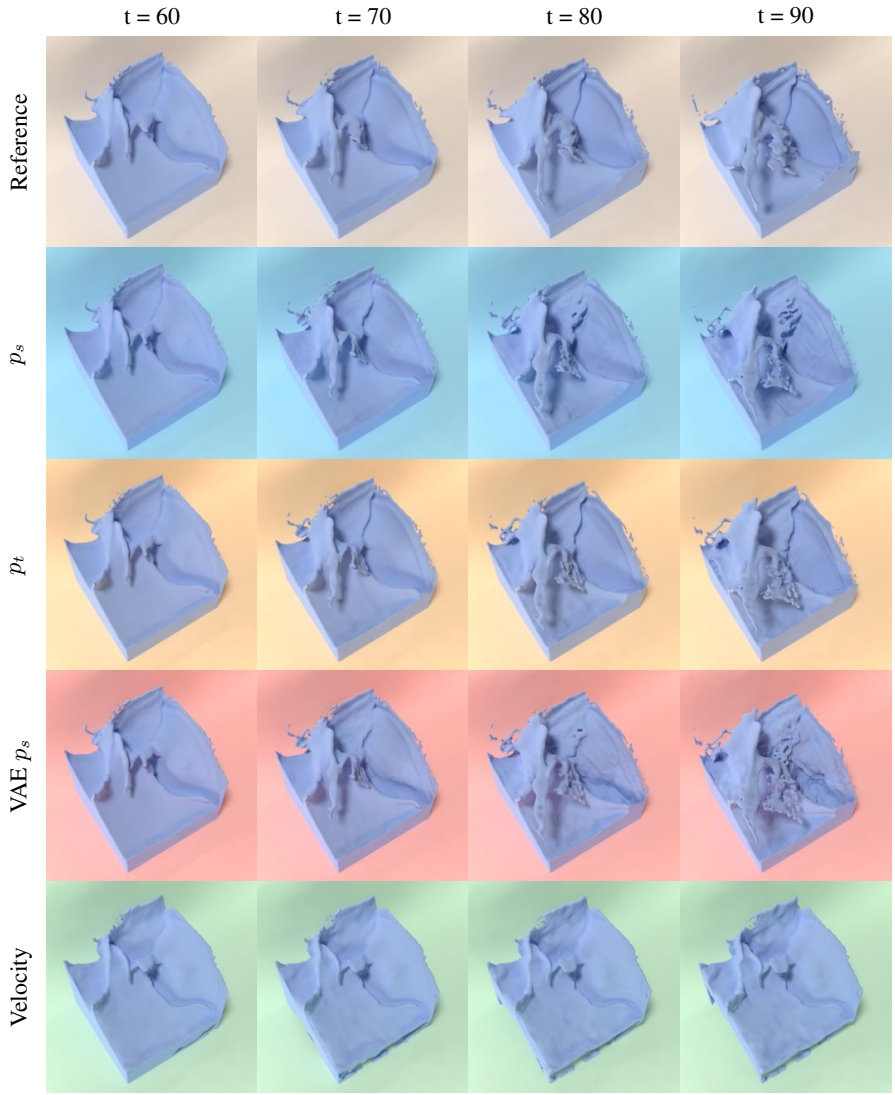
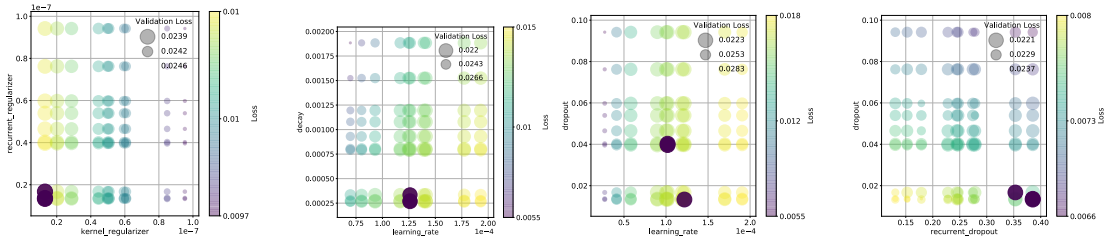


Figure 18: Additional comparison of liquid surfaces predicted by different architectures for 40 time steps for $i_p = \infty$, with prediction starting at time step 50. While the velocity version (green) leads to large errors in surface position, all three pressure versions closely capture the large scale motions. On smaller scales, both split pressure and especially VAE introduce artifacts.

E 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. 4.1.



(a) kernel regularizer and recurrent regularizer

(b) learning rate and learning rate decay

(c) learning rate and dropout

(d) recurrent dropout and dropout

Figure 19: Random search of hyperparameters for the prediction network

In Fig. 19 (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 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. 19a shows that the network performed best without any weight decay regularization applied.

Choosing good parameters leads to robust learning behavior in the training process, an example is shown in Fig. 20. 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.

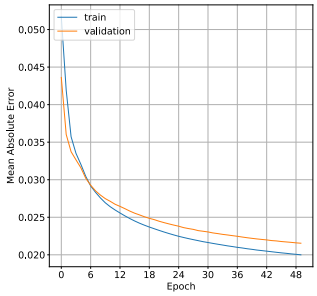


Figure 20: Training history of the *liquid128* p_t network.