A Model Details

In the decoder model, we use a similar network architecture as the NeRF paper [19]. As shown in Figure 8, we send a 3D point $\boldsymbol{x} \in \mathbb{R}^3$, a camera ray $\boldsymbol{d} \in \mathbb{R}^3$, and a state feature representation $\mathbf{s}_t$ into a fully-connected network and output the corresponding density σ_t and RGB color $\boldsymbol{c}_t$.

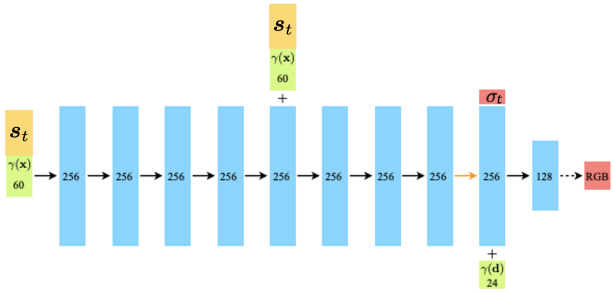


Figure 8: **A visualization of our decoder network architecture.** All layers are standard fully-connected layers, where the black arrows indicate layers with ReLU activations, orange arrows indicate layers with no activation, dashed black arrows indicate layers with sigmoid activation, and "+" denotes vector concatenation. We concatenate the positional encoding of the input location $\gamma(\boldsymbol{x})$ and our learned state representation $\boldsymbol{s}_t$, and pass them through 8 fully-connected ReLU layers, each with 256 channels. We follow the NeRF [19] architecture and include a skip connection that concatenates this input to the fifth layer's activation. An additional layer outputs the volume density σ_t (which is rectified using a ReLU to ensure that the output volume density is nonnegative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction $\gamma(\boldsymbol{d})$, and is processed by an additional fully-connected ReLU layer with 128 channels. A final layer (with a sigmoid activation) outputs the emitted RGB radiance at position $\boldsymbol{x}$, as viewed by a ray with direction $\boldsymbol{d}$, given the current 3D state representation $\boldsymbol{s}_t$.

B Environment Details

In the **FluidPour** environment, we generated $1,000$ trajectories for training. Each trajectory has 300 frames with 20 camera views sampled around the objects with a fixed distance towards the world origin. The action space for the control task is the position and tilting angle of the cup, which are randomly generated when constructing the training set.

In the **FluidShake** environment, we generated $1,000$ trajectories for training. Each trajectory has 300 frames with 20 camera views sampled around the objects with a fixed distance towards the world origin. The action space for the control task is the 2D location of the container in the world coordinate, which is also randomly generated when constructing the training set.

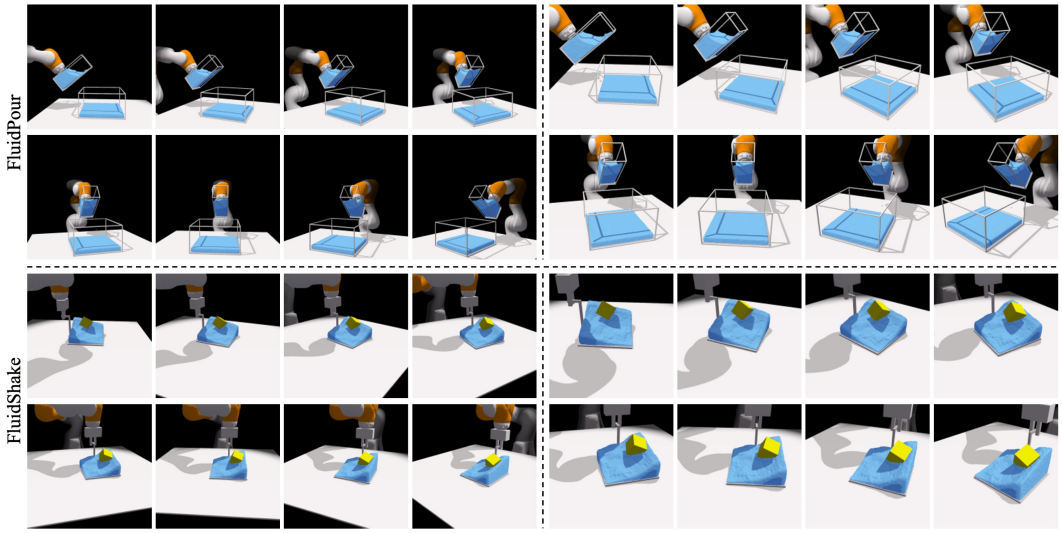
Figure 9 shows some example visual observations for FluidPour and FluidShake used during training. We also include some example images from viewpoints outside the training distribution, which are then used to evaluate our model's extrapolated generalization ability.

In **RigidStack**, we generated 800 trajectories for training. Each trajectory has 80 frames with 20 camera views sampled around the objects with a fixed distance towards the world origin.

In **RigidDrop**, we generated $1,000$ trajectories for training. Each trajectory has 50 frames with 20 camera views sampled around the objects with a fixed distance towards the world origin.

C Training Details

For the encoder and decoder model described in Figure 2, we use the Adam optimizer with the initial learning rate $5e^{-4}$ and decreased to $5e^{-5}$ for all the experiments. The batch size is 2. The hyperparameters in the decoder are the same as the original NeRF model [19] except the near and far distance between the objects and cameras are different in our environments. In our FluidPour environment, we have near $= 2.0$ and far $= 9.5$. In our FluidShake environment, we have near $= 2.0$ and far $= 7.0$. In our RigidStack environment, we have near $= 2.0$ and far $= 7.0$. In our FluidPour environment, we have near $= 2.0$ and far $= 6.0$.



(a) Example viewpoints used for training

(b) Example viewpoints outside training distribution used for goal specification in control

Figure 9: **Comparison between the viewpoints used for training and the subsequent viewpoint extrapolation experiments.** (a) We show some example images that the model used during training. For both environments, the camera is placed from a fixed distance and facing towards the world origin. (b) To evaluate the model's ability for viewpoint extrapolation, i.e., processing visual observations from viewpoints that are outside the training distribution, we generate another set of viewpoints that are closer, higher, and facing more downwards. It is clear from the figure that the images from viewpoints used during training are very different from the ones used for viewpoint extrapolation when measured using pixel difference. Therefore, it is essential to build a model that can directly reason over 3D to provide the desired extrapolation generalization ability. Although the model has access to visual observations from multiple cameras during training, it can only observe the environment from one camera when performing the downstream control task.

D Control Details

As discussed in Section 4.1, we use model-predictive path integral (MPPI) [57] to solve the MPC problem. MPPI is a sampling-based, gradient-free optimizer that considers temporal coordination between time steps when sampling action trajectories. At time t, the algorithm first samples M action sequences based on the current actions $\boldsymbol{a}_t, \ldots, \boldsymbol{a}_{T-1}$ via $\hat{\boldsymbol{a}}_h^k = \boldsymbol{a}_h + \boldsymbol{n}_h^k, k \in \{1, \ldots, M\}, h \in \{t, \ldots, T-1\}$. Each noise sample $\boldsymbol{n}_h^k$, denoting the noise value at the h^{th} time step of the k^{th} trajectory, is generated using filtering coefficient β as the following:

$$\begin{aligned} &\boldsymbol{u}_h^k \sim \mathcal{N}(0, \Sigma) \quad \forall k \in \{1, \ldots, M\}, h \in \{t, \ldots, T-1\} \\ &\boldsymbol{n}_h^k = \beta \cdot \boldsymbol{u}_h^k + (1-\beta) \cdot \boldsymbol{n}_{h-1}^k, \text{where } \boldsymbol{n}_{h<t}^k = 0. \end{aligned} \tag{5}$$

We then roll them out in parallel using the learned model on the GPU to derive $\hat{\boldsymbol{s}}_T^k, k \in \{1, \ldots, M\}$, and then re-weight the trajectories according to the reward to update the action sequence using a reward-weighting factor γ: $\boldsymbol{a}_h = (\sum_{k=1}^M \exp(\gamma \cdot R^k) \cdot \hat{\boldsymbol{a}}_h^k) / (\sum_{k=1}^M \exp(\gamma \cdot R^k)), h \in \{t, \ldots, T-1\}$, where $R^k = -\|\hat{\boldsymbol{s}}_T^k - \boldsymbol{s}^{\text{goal}}\|_2^2$. This procedure is repeated for L iterations at which point the best action sequence is selected.

The number of updating iteration for the auto-decoding test-time optimization K is 500. The number of sampled trajectories M during MPPI optimization is set to $1,000$. The number of iterations L for updating the action sequence is set to 100 for the first time step, and 10 for the subsequent control steps to maintain a better trade-off between efficiency and effectiveness. The reward-weighting factor γ is set to 50 and the filtering coefficient β is specified as 0.7. The control horizon T is set as 80 both for FluidPour and FluidShake. The hyperparameters are the same for all compared methods.

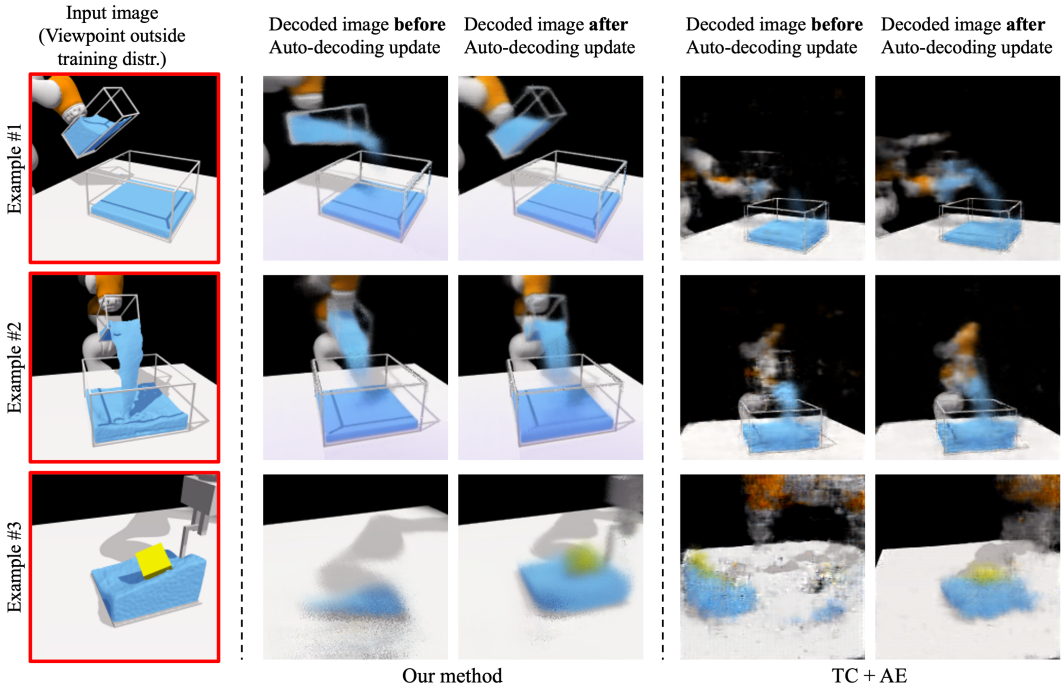


Figure 10: **Qualitative results on auto-decoding test-time optimization.** Following from the pipeline illustrated in Figure 3b, if the input image I_t is outside the training distribution as shown on the left column, the encoder won't be able to generate the most accurate state representation. When passing the predicted state embedding $\boldsymbol{s}_t$ and the same viewpoint as the input to the decoder, the generated image does not match the underlying scene as shown in the second column. We then calculate the L2 distance of the pixels between the generated image and true observation, backpropagating the gradient until the state representation and making updates to $\boldsymbol{s}_t$ using SGD. As discussed in Section 4.2, the translational equivariance nature of the decoder allows it to effectively optimize the latent representation to make it better reflect the 3D contents in the scene. After the optimization, the generated visual observation is much closer to the ground truth, as shown in the third column. On the contrary, the vanilla autoencoder that uses a CNN-based decoder won't be able to capture the underlying scene even with test-time auto-decoding optimization, as shown on the right.

E Additional Experimental Results

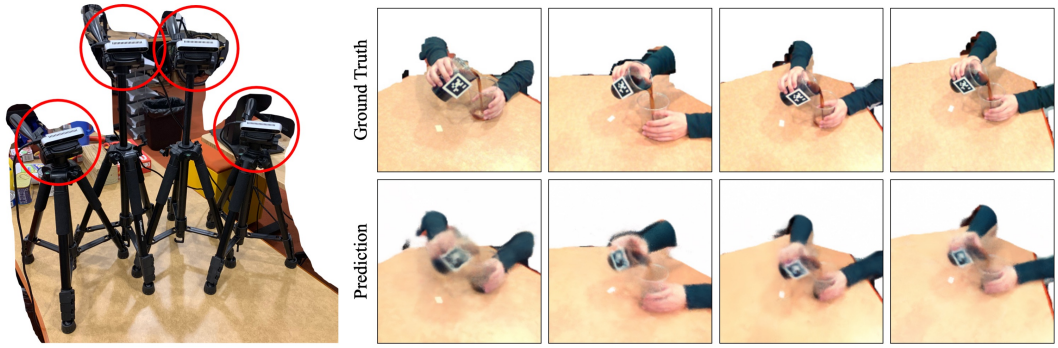
E.1 Auto-Decoding Test-Time Optimization for Viewpoint Extrapolation

Figure 10 shows the qualitative results on auto-decoding test-time optimization. When encountering an image from a viewpoint outside the training distribution, this mechanism can help us derive a better representation of the scene that holds a more accurate description of the 3D contents. The obtained representation after the optimization can then be used as the goal embedding $\boldsymbol{s}^{\text{goal}}$ in Equation 4 that the agent needs to achieve.

E.2 Validation of the Dynamics Prediction on Real-World Data

We further evaluate our model's dynamics prediction ability by conducting experiments on real-world data. As shown in Figure 11, we use four D415 RGBD cameras to record a human subject pouring water from one cup to another. The cameras are calibrated and synchronized with the frequency of 15 Hz. We recorded 50 pouring episodes of length 15 seconds, resulting in a dataset of 43,400 frames. We use the first 45 episodes for training and the remaining for testing.

To obtain the action, i.e., the movement of the cup that pours water out, we attached an AprilTag [59] on the cup to obtain the 6 DoF pose of the cup at each time step. From the supplementary video, you can see that our model can make open-loop future predictions on the testing trajectories in the representations space, i.e., given a latent scene representation of the current time step, the subsequent action sequence, our dynamic model can accurately predict the evolution of the latent scene representation and render the corresponding frames that closely resemble the ground truth.



(a) Data capture setup (four D415 cameras) (b) Comparison between our model's prediction and the ground truth

Figure 11: **Dynamics prediction using real-world data.** (a) We build a data recording set up containing four D415 RGBD cameras to record a human subject pouring water from one cup to another and then use the recorded data to evaluate our model's dynamics prediction ability. (b) We show a side-by-side comparison between the ground truth data and our model's prediction when predicting the future from the four camera views. Our model correctly identifies when the fluids pour out and start to fill the bottom container. Please see our supplementary video for better visualizations.

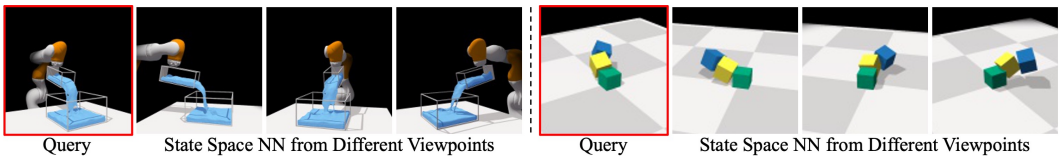


Figure 12: **Nearest neighbor (NN) results using our learned state representation.** Given a query image (red boundary), we search its nearest neighbors based on their state representation. Our learned scene representations can retrieve reasonable neighbor images, indicating that our state representations retain a good estimation of the contents inside the 3D scene and are invariant to camera poses.

E.3 Comparison With a PID Baseline That Only Matches the Robot's State

To show the necessity of modeling the fluid dynamics in our task, we include an additional baseline that uses PID control [60] to reach the robot state in the target image without worrying about the fluid. Note that this baseline uses additional information, including the ground truth state of the robot in both the current and the target image.

Our supplementary video shows a controlling trial where the goal is to leave a small amount of fluid in the container at the end of the control episode. Naively matching the container's position and orientation will not work, as shown in the PID baseline that the top container did not pour any fluid into the bottom container. In contrast, our model first learns to pour a certain amount of fluid out and then tilt the container back to match the target configuration. We further use the Chamfer distance to measure the models' performance in matching the fluid shape in the 3D point space, where our method significantly outperforms the PID baseline (Ours: 0.048237 vs. PID: 0.085727).

E.4 Nearest Neighbor Search Using the Learned Representations

When the goal image is specified from a viewpoint different from the agent's view, to ensure the planning problem defined in Equation 4 still work, it is essential that the distance in the learned feature space reflects the distance in the actual 3D space, i.e., scenes that are more similar in the real 3D space should be closer in the learned feature space, even if the visual observations are captured from different viewpoints. We visualize the nearest neighbor results in Figure 12. Given a query image, we search its nearest neighbors based on their state representation s (introduced in Section 3.1). Even if the images look quite different from each other when measured in pixel difference, the learned 3D-aware scene representations can retrieve reasonable neighborhood images that share similar 3D contents, indicating that the learned 3D-aware scene representations hold a good understanding of the real 3D scene and are invariant to viewpoint variations.

We conducted additional experiments to quantitatively measure the accuracy in finding the nearest neighbors (NN) across viewpoints of our proposed method. The results are shown in Table 1.

Env	Ours	NN in pixel space	Random
FluidPour	**1.772718**	147.971993	99.903261
FluidShake	**2.584928**	132.467998	99.646617

Table 1: **Quantitative results on nearest neighbor (NN) search from different viewpoints.** We calculate the accuracy of finding NN using the L1 distance between the ground truth and retrieved time indexes. Our method measures the distance in the learned representation space, which delivers the best performance, and the retrieved frame is, on average, around two time steps away from the ground truth.

Specifically, for each query frame, the model is asked to find the closest frame from a randomly selected viewpoint from a trajectory with 300 frames. We measure the accuracy using the L1 distance between the time indexes. Randomly selecting the closest frame leads to an averaged distance of 100 times steps between the selected frame and the ground truth frame. Selecting based on the pixel difference is even worse. Our method can accurately find the nearest neighbor. The average time step difference between the selected frame and the ground truth frame is 1.772718 in FluidPour and 2.584928 in FluidShake.

F Limitations and Future Works

Similar to many other data-driven methods, our model can deliver reasonable performance in regions well-supported by the data. However, for cases that our model has never seen before, e.g., more containers than during training, we wouldn't expect our model to generalize. Potential solutions may include (1) adding the examples and increasing the diversity of the training set or (2) using a more structured/compositional representation space instead of a single vector as in our model, which we leave for future works.