Latent Space Simulation for Carbon Capture Design Optimization

Brian Bartoldson,[1] Rui Wang,[1,2] Yucheng Fu,[3] David Widemann,[1]
Sam Nguyen,[1] Jie Bao,[3] Zhijie Xu,[3] Brenda Ng[1]

[1]Lawrence Livermore National Laboratory
[2]UCSD
[3]Pacific Northwest National Laboratory
{bartoldson, widemann1, nguyen116, ng30}@llnl.gov; ruw020@ucsd.edu; {yucheng.fu, jie.bao, zhijie.xu}@pnnl.gov

Abstract

The CO_2 capture efficiency in solvent-based carbon capture systems (CCSs) critically depends on the gas-solvent interfacial area (IA), making maximization of IA a foundational challenge in CCS design. While the IA associated with a particular CCS design can be estimated via a computational fluid dynamics (CFD) simulation, using CFD to derive the IAs associated with numerous CCS designs is prohibitively costly. Fortunately, previous works such as Deep Fluids (DF) (Kim et al., 2019) show that large simulation speedups are achievable by replacing CFD simulators with neural network (NN) surrogates that faithfully mimic the CFD simulation process. This raises the possibility of a fast, accurate replacement for a CFD simulator and therefore efficient approximation of the IAs required by CCS design optimization. Thus, here, we build on the DF approach to develop surrogates that can successfully be applied to our complex carbon-capture CFD simulations. Our optimized DF-style surrogates produce large speedups (4000x) while obtaining IA relative errors as low as 4% on unseen CCS configurations that lie within the range of training configurations. This hints at the promise of NN surrogates for our CCS design optimization problem. Nonetheless, DF has inherent limitations with respect to CCS design (e.g., limited transferability of trained models to new CCS packings). We conclude with ideas to address these challenges.

Introduction

Reducing the carbon intensity of electricity generation, the leading contributor of global greenhouse gas (GHG) emissions, is a key component of work towards achieving stabilization of GHG concentrations at a safe level (Edenhofer 2015). Critically, GHG emissions of fossil-based power plants can be reduced via carbon dioxide (CO_2) capture technologies (Edenhofer 2015).

Capture of CO_2 from power-plant flue gas is generally achieved through pre-combustion, oxyfuel-combustion, or post-combustion technologies (Koytsoumpa, Bergins, and Kakaras 2018). Of these, the most widely adopted is the solvent-based post-combustion approach (Koytsoumpa, Bergins, and Kakaras 2018; Wang et al. 2017), wherein CO_2 is captured through an absorption process caused by interaction between a particular liquid solvent and the flue gas inside a reactor column filled with packings (see Figure 1a).

A foundational challenge underlying design of such solvent-based carbon capture systems (CCSs) is optimizing the pairing of solvent and packing to maximize the gas-solvent interfacial area (IA) a for the CO_2 absorption reaction. Notably, CO_2 capture efficiency is determined by $k_g a$, where k_g is the gas film mass transfer coefficient (Singh, Galvin, and Sun 2017; Song, Seibert, and Rochelle 2018).

Capturing local effects from the hydrodynamics, heat, and mass transfer of the absorption process, CFD simulation helps us understand the IA associated with a solvent and packing configuration. While physically faithful, CFD is too computationally expensive to be used to evaluate potential configurations inside a CCS design optimization process.

We address this challenge by learning neural network (NN) surrogates for our CFD simulator. Specifically, we apply the Deep Fluids (Kim et al. 2019) surrogate approach and variants we introduce to CFD-simulated CCS volume fraction fields (from which we can compute IA). Once trained, these surrogates can quickly simulate the volume fraction fields of an unseen CCS configuration, enabling fast IA predictions for our CCS design optimization process.

Critically, our experiments focus on whether these fast surrogates accurately predict IA and liquid volume fraction.

While the original Deep Fluids (DF) approach gave visually realistic results on simpler problems, it produced high errors on our CCS application's complex domain. Thus, we developed multiple innovations to customize DF (e.g., end-to-end training and new NNs for latent-space simulation). Our innovations led to a final DF-inspired surrogate that can simulate CCSs with unseen configurations and predict IA with 4% relative error compared to the CFD simulation that is 4000x slower. This result demonstrates the promise of NN-based simulation for CCS design optimization.

Relevance of AI to Application

Modeling the spatiotemporal dynamics of a system is essential not just to our CCS application, but to a wide array of applications in physics, epidemiology, and molecular dynamics. Traditional approaches rely on running numerical simulations: known physical laws encoded in systems of complicated coupled differential equations are solved over space and time via numerical differentiation and integration schemes. However, such methods are computationally intensive, requiring expertise and manual engineering in each ap-

plication (Houska et al. 2012; J.C.Butcher 1996).

Consequently, there is an active, expansive literature on deep learning (DL) methods for accelerating or replacing numerical simulations (Wang 2021; Willard et al. 2020). For example, deep dynamics models can approximate high-dimensional spatiotemporal dynamics by directly forecasting future states, bypassing numerical integration (Wang et al. 2020; Wang, Walters, and Yu 2021; de Bezenac, Pajot, and Gallinari 2018). Sanchez-Gonzalez et al. (2020) designed a deep encoder-processor-decoder graph architecture for simulating fluid dynamics under a Lagrangian description. de Avila Belbute-Peres, Economon, and Kolter (2020) combined graph neural networks with a CFD simulator run on a coarse mesh to generate high-resolution fluid flow predictions. Tompson et al. (2017) replaced the numerical pressure solver with convolutional networks in Eulerian fluid simulation and obtained realistic results.

When a CFD model is replaced with a surrogate that forecasts future states within a lower-dimensional latent space, larger speedups are realizable. These "latent space physics" can be learned with an autoencoder that maps physical fields to latent vectors and an LSTM that models the temporal evolution of the latent vectors. These surrogates enable fluid flow pressure field simulations hundreds of times faster than CFD's (Wiewel, Becher, and Thuerey 2019). In Deep Fluids (DF) (Kim et al. 2019), the LSTM is replaced by an MLP, and an auxiliary term is added to the loss to ensure divergence-free motion for incompressible flows.

The speedup provided by these methods is relevant to our CCS design optimization, which requires volume fraction field simulations for numerous CCS configurations. Accordingly, we applied DF to our CCS data. Our innovations for DF include new latent-space simulation networks—we explore various MLPs, CNNs, LSTMs (Hochreiter and Schmidhuber 1997), and transformers (Vaswani et al. 2017)—as well as end-to-end training.

Application and AI Approach

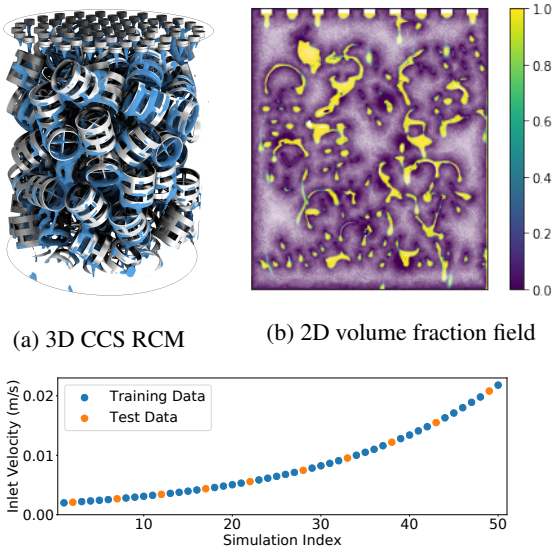
This work augments DF to achieve acceptable surrogate predictions for our CO_2-capture-simulation application. We first describe our application and its related CFD data. Then, we discuss our DF-inspired AI surrogates and the innovations we developed to improve DF's performance.

Application and CFD Data Description

Our application is IA-based design optimization of CCSs. In particular, we seek a means to find CCSs associated with high IA, a key determinant of CO_2-capture efficiency. A typical (but slow) approach is to predict IA using the volume fraction fields resulting from a CFD simulation of a *representative column model* (RCM) (Figure 1a), which is a representative section of the bench-scale column of our CCS.

This involves simulating the counter-current solvent and gas flow in the 3D RCM (containing 6.5 million data points) then solving continuity and momentum equations with the Volume of Fluid (VOF) method (Fu et al. 2020):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0, \tag{1}$$



(a) 3D CCS RCM

(b) 2D volume fraction field

(c) Solvent inlet velocity v for each CFD simulation.

Figure 1: (a) The 3D RCM of our CCS. In our simulation, parameter v is the rate at which CO_2BOL (solvent) is injected into the column from above, while CO_2-laden gas is injected from below. (b) Volume fraction field in 2D domain at $t = 500$ and $v = 0.01\,\mathrm{m/s}$. (c) The 50 CCS configurations we used to train and test our DF-inspired surrogate.

$$\frac{\partial(\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g} + \mathbf{F}_\sigma, \tag{2}$$

where ρ is density, μ is viscosity, $\mathbf{u}$ is velocity, p is pressure, $\mathbf{g}$ is gravity, and $\mathbf{F}_\sigma$ is the surface tension arising at the gas-liquid interface. The density and viscosity are calculated by a volume fraction average of liquid (α) and gas phase (1-α). The evolution of α is governed by the transport equation:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{u}\alpha) = 0. \tag{3}$$

Running many such CFD simulations to thoroughly explore the CCS configuration space (which involves: packing type, e.g., Gyroid vs. Schwarz-D; inlet velocity v; solvent viscosity/surface-tension; etc.) is computationally infeasible. However, it may be possible to use existing CFD data to train an AI surrogate model to quickly and accurately simulate dynamics of CCSs with never-before-seen configuration settings. During IA-based optimization, this surrogate's faster speed would allow more thorough exploration of the CCS configuration space, improving the chance of finding a CCS design with better CO_2-capture performance.

To evaluate the utility of AI surrogates for our application, we measure the speed and accuracy of surrogate simulations of CCSs with values of the solvent inlet velocity parameter not seen during training. Our data is $N = 50$ CCS CFD simulations, each with a unique inlet velocity (Figure 1c). $N_{train} = 40$ simulations are used for training, while the remaining $N_{test} = 10$ simulations are for evaluation.

As a first step, we worked with CFD data from a 2D domain, derived from a vertical slice of the larger 3D CCS do-

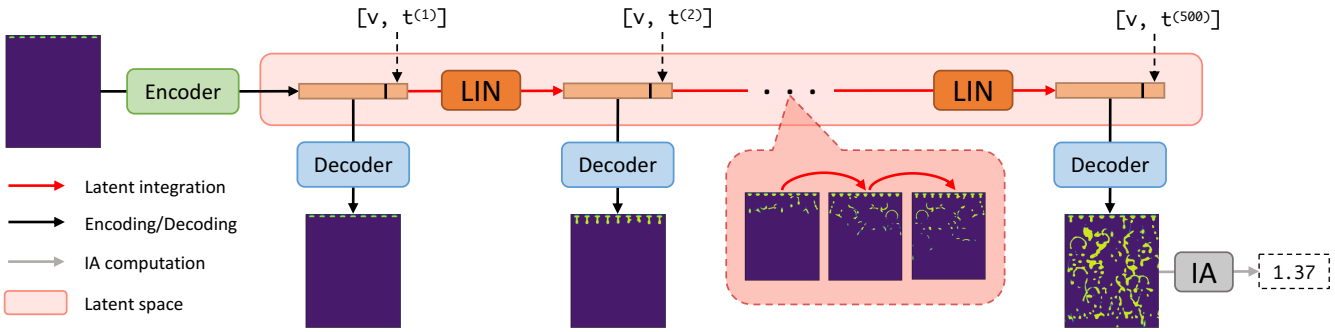


Figure 2: DF surrogate simulation ("full rollout", $w = T-1$) of CCS's 2D volume fraction field, with IA computation at $t = T$.

main (scaling to the 3D case is discussed in our path to deployment section). Like the 3D domain, the 2D domain with 150,073 polygonal mesh grid points has its counter-current flow solved via the VOF method. Starting from an initial state, a particular CCS configuration on this 2D domain reaches a steady state in 5000 timesteps (Figure 1b shows a single timestep), using a wall-clock time $W_{\text{CFD}} = 3600\,\text{s}$ when run on 96 CPU cores.

Before training surrogates, the simulation data is temporally downsampled to 500 timesteps. As our AI approach uses CNNs that operate on uniform grids, the simulation data is then spatially downsampled via linear interpolation to obtain uniform grid data $g_t^{(i)} \in G$, where $G = \mathbb{R}^{k\times k}$, $i \in 1, 2, ..., 50$ denotes the simulation index, and $t \in \{1, 2, ..., T\}$ is the timestep with $T = 500$ (we omit i and t when referring to generic elements of G). We report results for $k = 64$ but saw similar performance with $k = 128$.

Note that we train surrogate models only on volume-fraction field dynamics because our application is focused on IA, which is a function of the volume-fraction field $g_t^{(i)}$.

AI Approach to Our CFD Simulations

Here, we discuss surrogate simulation with DF and our innovations. DF is a data-driven, NN-based, non-intrusive surrogate approach. In our application, surrogate input is the timestep $t \in \mathbb{N}$, a *frame* $g_t \in G$ that represents the volume fraction field at that timestep, and the inlet velocity $v \in \mathbb{R}$ of the CCS. Surrogate outputs $\widehat{g}_{t+j}$ are volume fraction fields at times $t + j$. (Note, we use hats to denote predicted quantities.) DF achieves this as follows (see Figure 2):

1. **Encode the frame:** Reduce the dimensionality of the frame g_t via an *encoder network* $E : G \to \mathbb{R}^c$. The output of E is a *latent vector* denoted by l_t.
2. **Add simulation-configuration details to latent vector:** Replace the last two elements of $l_t = E(g_t)$ with $[v, t]$.
3. **Temporally advance the latent vector:** Use a separate *latent integration network* (LIN) $I : \mathbb{R}^c \to \mathbb{R}^c$ to predict the change $\Delta l_t = l_{t+1} - l_t$ and thereby obtain the next latent vector $\widehat{l}_{t+1} = l_t + I(l_t)$; perform this step w times.
4. **Decode the new latent vector(s):** Feed $\widehat{l}_{t+j}$ to the *decoder network* $D : \mathbb{R}^c \to G$ to obtain $\widehat{g}_{t+j}$.

E and D are trained together as an autoencoder, more generally as a latent vector model LVM : $G \to G \times \mathbb{R}^c$, which encodes then decodes a g_t and provides the corresponding latent vector (or encoding) l_t as output. Notably, by inserting the simulation-specific information $[v, t]$ into the encoder's latent code rather than predicting it, we remove the supervised learning of these values done in (Kim et al. 2019) in exchange for allowing the decoder and LIN to train with the exact simulation parameters $[v, t]$. Given a trained LVM, the DF approach computes the latent vector l for every training simulation frame g. The LIN I is then trained to predict the deltas among these l within a *rollout* window ($\Delta l_\tau, \tau \in \{t, ..., t + w - 1\}$ with w as the window size) given only l_t (and thus also $[v, t]$) as input.

Once I is trained, it can be iteratively applied to perform a full rollout that produces $\widehat{l}_t$ and $\widehat{g}_t$ for $t \in \{1, ..., T\}$, given only g_1 and v as input. Full rollouts are useful when using trained surrogates to replace CFD simulation (see Figure 2).

In our experiments, we study how surrogate performance changes in response to modifying the training-time rollout window size w and other variables. Additionally, we evaluate larger modifications to the DF approach: joint training of the LVM and LIN (end-to-end training), and new LVM/LIN architectures (e.g., transformers).

Latent vector models Like the original DF approach, we use a CNN autoencoder (comprising both E and D) as an LVM. The encoder network E is four Conv2D layers (with stride 2, padding 1, kernel size 3, and layer-specific channel counts $[128, 256, 512, 1024]$) followed by a linear layer that maps from the final convolutional feature map to $\mathbb{R}^c$. The decoder network D completes a symmetric architecture for the LVM, with a linear layer followed by four ConvTranspose2D layers (using the layer-specific channel counts $[512, 256, 128, 1]$) so that the output has g's shape.

We also consider replacing the CNN autoencoder with an LVM derived from an SVD of the training data ($B = U\Sigma V^\top$, where $B \in \mathbb{R}^{k^2 \times N_{train}T}$). Specifically, encoding then decoding B via multiplication with the truncated matrix $U_{:,1:c}$ gives the best rank-c approximation $\hat{B}$ (Eckart and Young 1936). This provides a strong baseline for the larger and slower CNN LVM, which also compresses frames to size c but uses nonlinear transformations to make possible decodings $\hat{B}$ with rank $m{>}c$ and thus smaller error.

Finally, we explore the usage of vision transformers (ViTs) as LVMs. As in Dosovitskiy et al. (2021), we split each input frame into 16×16 patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard transformer encoder. Then, the transformer decoder takes as input the latent vectors from the encoder and an extra learnable embedding. The learnable embedding's state at the output of the transformer decoder serves as the frame representation and is linearly transformed to generate the reconstructed frame.

Latent integration networks As in DF, we use an MLP to learn I, the latent integration network (LIN). We explore several modifications to the proposed MLP LIN. We modify the LIN's depth and width, and augment the input dimension to exploit more history (i.e., more frames as input) such that $I : \mathbb{R}^{s \times c} \rightarrow \mathbb{R}^c$, where sequence length s is the number of prior timesteps the LIN uses as input (when $t < s$, the $s - t$ remaining rows/timesteps of the input space are initialized at $\mathbf{0}$ and filled gradually as the LIN produces them).

In addition to MLPs, we evaluated other competitive sequence models, such as CNNs (Oord et al. 2016), LSTMs (Hochreiter and Schmidhuber 1997), and transformers (Vaswani et al. 2017), as LINs. For the LSTM and transformer networks, we use PyTorch implementations: the transformer is a series of six 8-head transformer encoder layers with hidden dimension c, while the LSTM network is a series of LSTM layers with hidden dimensions $H = [h_1, ..., h_n]$. To test CNNs, we created AutoRegressive CNN (ARC) LINs that comprise 1D convolutional layers with output channel counts $[h_1, h_2, ..., h_n, c]$, the first layer using c input channels, and all layers using kernel size 3 and padding 1. The output layer of each LIN is a linear layer that maps to $\mathbb{R}^c$, except for the ARC LIN's output layer, which uses a learnable linear combination of the rows from the final feature map to compute $\Delta l_t \in \mathbb{R}^c$. Finally, with the ViT LVM, we used a single linear layer for the LIN, as the performance bottleneck when using the ViT was the LVM.

Performance optimization and evaluation When training LVMs, we optimize the relative error loss $L_{\text{RE}}(\widehat{g}, g) = \frac{\|\widehat{g}-g\|_2}{\|g\|_2}$ averaged across the K volume fraction fields in a batch, as we found this performed better than RMSE and preliminary results showed no quantitative benefit of the gradient-matching loss term used in DF. Similarly, to train the LIN, we optimize the relative error loss $L_{\text{RE}}(\widehat{l}_t, l_t)$ averaged over the K w-timestep latent vector rollouts (i.e., there are Kw unique relative errors each batch). When using end-to-end training, the Kw rolled-out latent vectors are decoded to frames $\widehat{g}_t$, then $L_{\text{RE}}(\widehat{g}_t, g_t)$ is computed and gradients are taken with respect to the parameters of E, I, and D. We show the effect of training the LIN with RMSE (instead of L_{RE}) and of different values of w and s in our experiments.

While our loss functions aim to ensure accurate reconstruction of g_t, our ultimate goal is IA-based optimization of CCS designs. Thus, we are primarily interested in the ability of DF-inspired surrogates to estimate steady-state IA correctly, given only the initial conditions g_1 and inlet velocity v of a new, unseen CCS. After training

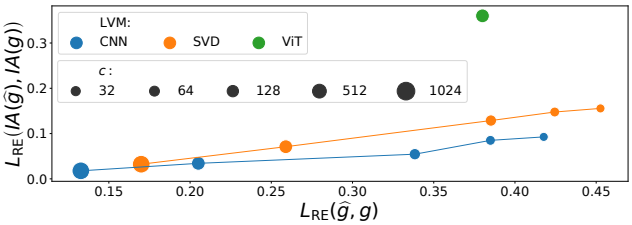


Figure 3: Mean test error for our LVMs at up to five c values.

our models, we measure the steady-state IA relative error $L_{\text{RE}}\left(IA(\widehat{g}_T^{(i)}), IA(g_T^{(i)})\right)$, where i indexes the N_{test} previously-unseen test simulations, $\widehat{g}_T^{(i)}$ is derived from a full rollout, and $IA : G \rightarrow \mathbb{R}$ is the integral of surface areas for which the volume fraction is 50% (on a 2D volume fraction field, such surfaces correspond to volume fraction isocurves).

Then, we report the mean error: $\text{Error}_{\text{IA}} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} L_{\text{RE}}\left(IA(\widehat{g}_T^{(i)}), IA(g_T^{(i)})\right)$.

While this mean, final-timestep IA relative error is a critical metric for our application, we are also interested in whether surrogates could potentially replace CFD simulations for purposes other than IA estimation. Thus, to test the ability of surrogates to mimic CFD volume fraction dynamics at all timesteps, we use $\widehat{g}_t^{(i)}$ from full rollouts on the test data to compute average volume fraction relative error: $\text{Error}_{\text{VF}} = \frac{1}{N_{test}T} \sum_{i=1}^{N_{test}} \sum_{t=1}^{T} L_{\text{RE}}\left(\widehat{g}_t^{(i)}, g_t^{(i)}\right)$.

Finally, we use the ratio of the wall-clock time of the CFD simulation to that of a surrogate's full rollout (which is run on four NVIDIA P100 GPUs) to compute the relative speedup offered by our AI approach: $S_W = \frac{W_{\text{CFD}}}{W_{\text{AI}}}$.

Experiments

To understand whether AI surrogates may help our IA-based CCS-design-optimization task, we use our CFD data to study the accuracy and speedups of DF surrogates and our innovative approaches. We report the mean of two runs (and standard deviation) for each result. Our code is available here: https://github.com/CCSI-Toolset/DeeperFluids.

The performance of DF-inspired surrogates depends on the performances of both LIN (I) and LVM (E and D). So we first study performance of LVMs. Given the best LVM, we use its latent vectors to train LINs, with which we perform "full rollouts" to evaluate performance of the entire surrogate system via Error_{VF}, Error_{IA}, and S_W. We then test the benefit of end-to-end training of the LVM and LIN. Finally, we study the effects of key model/training hyperparameters.

LVM Architecture Comparison

To test how the DF approach might benefit our application, we first consider LVM error of three model classes, including two new approaches that we introduced to DF. Specifically, we compare the performances of a CNN-based au-

LIN	Error_{IA}	Error_{VF}	S_W
ARC	0.07 (0.00)	0.49 (0.00)	4800
LSTM	0.08 (0.01)	0.49 (0.01)	2700
MLP	**0.06** (0.00)	**0.47** (0.00)	5400
Transformer	0.08 (0.04)	0.51 (0.00)	4300

Table 1: For each LIN architecture, the best Error_{IA} and Error_{VF} found via hyperparameter sweeps (standard deviations in parentheses) with wall-clock time speedup S_W.

toencoder (used in DF); a ViT (Dosovitskiy et al. 2021) with similar parameter count; and SVD, a faster, linear approach.

To measure performance, we average the frame reconstruction and IA errors, $L_{\text{RE}}(\widehat{g}, g)$ and $L_{\text{RE}}(IA(\widehat{g}_T), IA(g_T))$, across the test data. As SVD is linear, we expect it to be less accurate than CNNs/ViTs, which are powerful approximators that can represent nonlinear functions. However, ViTs and CNNs require careful optimization to perform well. This is especially true for ViTs, which lack the inductive biases of CNNs (e.g., translation equivariance and locality), harming their performance when data is scarce (Dosovitskiy et al. 2021) as it is here. Thus, SVD may potentially outperform CNNs or ViTs. Consistent with the difficulty of training ViTs without extra/sufficient data, we found SVD outperformed our ViT on the LVM reconstruction task (Figure 3). However, SVD did not outperform the CNN. The ability of CNNs but not ViTs to outperform SVD reinforces the (particularly important in our low-data regime) point that image-processing inductive biases provided by convolutional layers offer a significant benefit. As **CNN-based autoencoders produced the best LVMs**, we use them in our remaining experiments except where we indicate use of a ViT LVM.

LIN Architecture Comparison

Given the latent vectors from the CNN LVM, we sought to determine the best LIN architecture. While the original DF used an MLP for I, other architectures such as LSTM, CNN, and transformer offer competitive performance in sequence modeling. To test whether these other architectures may outperform MLPs on our data, we trained each architecture using several settings for c, s, w, b, H, and the loss function. Then, with this set of trained models, we compared the best Error_{VF} and Error_{IA} of each architecture.

While an MLP had the best performance across multiple runs (0.06 mean Error_{IA}), we found that **the best single model for our IA-based optimization application was a transformer**, with Error_{IA} of just 0.04 (Table 1). Each architecture in Table 1 has roughly 1×10^5 parameters. For each performance measurement, the best models were found by optimizing L_{RE} rather than RMSE, except for the best transformer according to Error_{IA}, which was found by optimizing RMSE. In addition, the best models trained with $s = 6$, except for the the best transformer and MLP models according to Error_{IA}, which trained with $s = 1$. Finally, all models attained their best performance with $50 \leq w \leq 300$.

The surrogate speedups are significant (up to 5400x) and partly stem from their 10x larger timesteps.

Surrogate	Training style	Error_{IA}	Error_{VF}
CNN+ARC	Classic DF	0.07 (0.00)	0.52 (0.01)
CNN+ARC	E2E DF	**0.06** (0.00)	**0.48** (0.00)
ViT+Linear	Classic DF	0.53 (0.25)	0.77 (0.08)
ViT+Linear	E2E DF	0.36 (0.08)	0.59 (0.03)

Table 2: The effect of end-to-end training/fine-tuning.

	Error_{IA}					
H	w=20	50	150	200	300	499
1024, 512	0.33 (0.14)	0.08 (0.00)	0.09 (0.00)	0.10 (0.00)	0.11 (0.00)	2.53 (0.26)
128, 128, 128	0.64 (0.11)	0.08 (0.01)	0.08 (0.00)	0.07 (0.00)	**0.06** (0.00)	2.25 (0.18)

Table 3: Effect on Error_{IA} of window w and layer sizes H.

End-to-End Training/Fine-tuning

The original DF trained LIN and LVM separately. Here, we investigate whether end-to-end (E2E) training improves performance. We applied E2E fine-tuning to a pretrained CNN LVM + ARC LIN surrogate and E2E training to a ViT LVM + linear LIN surrogate, then compared against classic training (Table 2). Consistent with the benefit of LIN training that directly optimizes $L_{\text{RE}}(D(\widehat{l}), g)$, we found both **E2E approaches significantly improved** Error_{VF} **and** Error_{IA}.

Hyperparameter Study

Here, we illustrate the effects of key DF hyperparameters, noting when optimal settings deviate from those in the original DF model. Settings of factors not shown in a table are held constant and may be found in our open-source code.

Deeper LINs and larger rollout windows have higher performance We found that, for our data, the wide MLP LIN with $H = [1024, 512]$ used in the original DF performed worse than a narrower, deeper MLP LIN with $H = [128, 128, 128]$ (Table 3). Since the narrower, deeper LIN offered superior performance with far fewer parameters, we used $H = [128, 128, 128]$ for our other experiments. Additionally, we found that increasing the window size, which allows the model to see and learn to control accumulation of error from repeated LIN applications during training, is helpful up to a point: as shown by the high error at $w = 499$, too many LIN applications make training difficult.

Relative error outperforms RMSE as a LIN loss function, longer sequence lengths can be helpful Optimizing the relative error loss (L_{RE}) of $\widehat{l}$ rather than the RMSE significantly improved Error_{VF} and led to the best Error_{IA} (Table 4). Use of L_{RE} as an optimization target may lead to better Error_{VF} because each of these functions computes a relative error; i.e., L_{RE} has an advantage over RMSE when high relative error in latent space translates to high relative

		Error_{IA}		Error_{VF}	
LIN	s	L_{RE}	RMSE	L_{RE}	RMSE
ARC	1	0.10 (0.02)	0.09 (0.00)	0.52 (0.01)	0.55 (0.00)
	6	0.12 (0.03)	0.12 (0.02)	0.52 (0.01)	0.53 (0.00)
LSTM	1	0.11 (0.01)	0.09 (0.02)	0.53 (0.03)	0.54 (0.00)
	6	0.11 (0.01)	0.10 (0.01)	0.52 (0.00)	0.54 (0.00)
MLP	1	**0.06** (0.00)	0.08 (0.00)	0.53 (0.00)	0.55 (0.00)
	6	0.09 (0.00)	0.10 (0.03)	**0.49** (0.00)	0.52 (0.00)
Transformer	1	0.09 (0.01)	0.08 (0.04)	0.52 (0.00)	0.55 (0.00)
	6	0.09 (0.03)	0.14 (0.08)	0.53 (0.00)	0.68 (0.11)

Table 4: The effect on Error_{IA} and Error_{VF} of LIN architecture, input sequence length s, and training loss.

error in frame space. The fact that L_{RE} also led to the best Error_{IA} may be attributable to the more accurate frame reconstructions obtained when using this loss.

Increasing the sequence length s tended to improve Error_{VF} (Table 4), regardless of optimization target (except with the transformer). Thus, for most LINs, to minimize discrepancies between CFD and AI surrogate simulations (Error_{VF}), our results support combining the relative error loss L_{RE} with $s > 1$. Notably, Kim et al. (2019) use RMSE and $s = 1$ (with the MLP LIN).

Path to Deployment

We now discuss the potential of our NN surrogate in the design of an industrial scale coal-fired power plant CCS and the necessary steps to achieve this deployment. At industrial scale, the 3D packed column for carbon capture is meters long, much larger than the 3D RCM shown in Figure 1a. To scale our approach to this case, two critical steps are required. First, the surrogate will be modified to predict the data generated from the 3D RCM CFD simulations. At this 3D RCM scale, the CFD simulation cost is manageable, so we can produce data as needed for training the NN surrogate. Second, after verifying the accuracy of the 3D surrogates, they will be further scaled to predict industrial scale column data. Notably, the NN is well suited for this upscaling due to its remarkably cost-effective performance compared to the physics-based CFD simulations. Further, modifying our LVMs to accommodate 3D data is straightforward, and the knowledge and methodologies relevant to the 2D models can be transferred to the 3D case to avoid costly methodology optimization at the 3D scale. However, we expect that the larger 3D domains will require more computing resources and data to reach acceptable performance levels.

Once trained, the 3D NN surrogates can provide data for various industry-scale CCS configurations via high throughput IA prediction. We can leverage this to perform IA-based CCS optimization by coupling the NNs with FOQUS (Eslick et al. 2014), an advanced optimization and uncertainty quantification tool developed for carbon capture simulations. Additionally, the NN surrogate-based IA prediction can be integrated into ASPEN (Van der Spek et al. 2020), which accounts for multiple systems in a coal-fired power plant to produce carbon capture techno-economic analysis.

Discussion and Limitations

IA-based design optimization of CCSs is prohibitively costly when relying on CFD, motivating our study of the DF surrogate simulation approach and variants we introduced here. We found these AI surrogates could quickly simulate volume fraction field dynamics accurately enough to make low-error IA predictions for new CCS configurations. Indeed, DF appears to be promising for our application, as it is 4000x as fast as our CFD model with just 4% IA error.

While several of the DF surrogate variants we explored neared this performance level, our results on 2D data suggest that the most promising surrogate employs a CNN LVM, a transformer LIN, and E2E training. Thus, this surrogate will be our focus as we begin scaling to and evaluating on 3D CCS CFD data, a crucial next step on our path to industrial-scale deployment of our DF-style surrogate models.

Importantly, though, DF has limitations for our application. The CNN LVM likely needs to be retrained to adapt to CCSs with different packings, as performance of CNNs on data distributions different from the train distribution significantly degrades (Hendrycks and Dietterich 2018; Beery, Van Horn, and Perona 2018). For example, an LVM could minimize velocity field error by learning to always output 0 at packing locations, but doing so would cause high error on new data with different packing locations. To guard against this, an interesting future direction is to use graph neural networks (GNNs) to learn dynamics from nodes on graphs that represent mesh-based simulation data. The GNN's learned internode dynamics have been shown to generalize well to new domains (Pfaff et al. 2020), and such generalization could allow us to change CCS packings without retraining.

Relatedly, our analysis left unclear whether performances observed on the data we studied would extend to models trained on new packing configurations or models tested on levels of v beyond those seen during training. Notably, our analysis covered most practical operating values of v. Further, a preliminary study found that models that perform well on the packing configuration used here can be trained on a new packing configuration and obtain similar performance. Thus, the DF approach could potentially be used to train a model for each packing configuration, allowing the design-optimization procedure to quickly obtain IA values at any relevant v via simulation with the model trained on the packing being considered (e.g., rather than using a single GNN model that works well for all packing configurations).

Another limitation of the DF approach is high field relative error. While errors of surrogate simulated fields were low enough to facilitate accurate IA estimates, similar applications may require more faithfulness to the CFD fields. Future work, akin to the end-to-end training we studied, could further improve relative errors of DF-predicted fields.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344. The authors acknowledge financial support through the Point Source Carbon Capture Program at the U.S. Department of

Energy, Office of Fossil Energy and Carbon Management. This work was conducted as part of the Carbon Capture Simulation for Industry Impact (CCSI2) project. We thank Phan Nguyen for helpful comments.

References

Beery, S.; Van Horn, G.; and Perona, P. 2018. Recognition in terra incognita. In *Proceedings of the European conference on computer vision (ECCV)*, 456–473.

de Avila Belbute-Peres, F.; Economon, T. D.; and Kolter, J. Z. 2020. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. *ArXiv*, abs/2007.04439.

de Bezenac, E.; Pajot, A.; and Gallinari, P. 2018. Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge. In *International Conference on Learning Representations*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Eckart, C.; and Young, G. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3).

Edenhofer, O. 2015. *Climate change 2014: mitigation of climate change*, volume 3. Cambridge University Press.

Eslick, J. C.; Ng, B.; Gao, Q.; Tong, C. H.; Sahinidis, N. V.; and Miller, D. C. 2014. A framework for optimization and quantification of uncertainty and sensitivity for developing carbon capture systems. *Energy Procedia*, 63: 1055–1063.

Fu, Y.; Bao, J.; Singh, R.; Wang, C.; and Xu, Z. 2020. Investigation of countercurrent flow profile and liquid holdup in random packed column with local CFD data. *Chemical Engineering Science*, 221: 115693.

Hendrycks, D.; and Dietterich, T. 2018. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *International Conference on Learning Representations*.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Houska, B.; Logist, F.; Diehl, M.; and Impe, J. V. 2012. A tutorial on numerical methods for state and parameter estimation in nonlinear dynamic systems. In Alberer, D.; Hjalmarsson, H.; and Re, L. D., eds., *Identification for Automotive Systems, Volume 418, Lecture Notes in Control and Information Sciences*, 67–88. Springer.

J.C.Butcher. 1996. *Applied Numerical Mathematics*, volume 20. Elsevier B.V.

Kim, B.; Azevedo, V. C.; Thuerey, N.; Kim, T.; Gross, M.; and Solenthaler, B. 2019. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, 59–70. Wiley Online Library.

Koytsoumpa, E. I.; Bergins, C.; and Kakaras, E. 2018. The CO2 economy: Review of CO2 capture and reuse technologies. *The Journal of Supercritical Fluids*, 132: 3–16.

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; and Battaglia, P. 2020. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*.

Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; and Battaglia, P. 2020. Learning to Simulate Complex Physics with Graph Networks. *arXiv preprint arXiv:2002.09405*.

Singh, R. K.; Galvin, J. E.; and Sun, X. 2017. Hydrodynamics of the rivulet flow over corrugated sheet used in structured packings. *International Journal of Greenhouse Gas Control*, 64: 87–98.

Song, D.; Seibert, A. F.; and Rochelle, G. T. 2018. Mass transfer parameters for packings: effect of viscosity. *Industrial & Engineering Chemistry Research*, 57(2): 718–729.

Tompson, J.; Schlachter, K.; Sprechmann, P.; and Perlin, K. 2017. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3424–3433.

Van der Spek, M.; Fout, T.; Garcia, M.; Kuncheekanna, V. N.; Matuszewski, M.; McCoy, S.; Morgan, J.; Nazir, S. M.; Ramirez, A.; Roussanaly, S.; et al. 2020. Uncertainty analysis in the techno-economic assessment of CO2 capture and storage technologies. Critical review and guidelines for use. *International Journal of Greenhouse Gas Control*, 100: 103113.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, R. 2021. Physics-Guided Deep Learning for Dynamical Systems: A survey. *arXiv preprint arXiv:2107.01272*.

Wang, R.; Kashinath, K.; Mustafa, M.; Albert, A.; and Yu, R. 2020. Towards Physics-informed Deep Learning for Turbulent Flow Prediction. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Wang, R.; Walters, R.; and Yu, R. 2021. Incorporating Symmetry into Deep Dynamics Models for Improved Generalization. In *International Conference on Learning Representations*.

Wang, Y.; Zhao, L.; Otto, A.; Robinius, M.; and Stolten, D. 2017. A review of post-combustion CO2 capture technologies from coal-fired power plants. *Energy Procedia*, 114: 650–665.

Wiewel, S.; Becher, M.; and Thuerey, N. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, volume 38, 71–82.

Willard, J.; Jia, X.; Xu, S.; Steinbach, M. S.; and Kumar, V. 2020. Integrating Physics-Based Modeling with Machine Learning: A Survey. *CoRR*, abs/2003.04919.