Supplementary Information for:

Encoding multiphase dynamics to predict spatiotemporal evolution via latent-space operators

Hongyuan Men[1,2,†], Yixuan Mao[3,†], V.L. Tagarielli[3], F. Montomoli[3], Hongwei Liu[1], Xinliang Li[1,2,*], and Menglan Duan[4,*]

[1]LHD, Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China
[2]School of Engineering Science, University of Chinese Academy of Sciences, Beijing 100049, China
[3]Department of Aeronautics, Imperial College London, London SW7 2AZ, UK
[4]Institute for Ocean Engineering, Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China
[†]Equally contributed
[*]Corresponding author: Xinliang Li (lixl@imech.ac.cn), Menglan Duan (mlduan@sz.tsinghua.edu.cn)

Contents

This supplementary document provides a detailed description of the proposed method, algorithm, examples, and discussion of technical challenges for modeling and discovery of multiphase dynamics.

A Method

A.1 Preliminaries

This study aims to predict the spatiotemporal evolution of phase volume-fraction fields at future times from their initial flow filed. We define the problem as follows: given the initial volume-fraction field $\mathcal{X}_0 : \mathbf{\Phi}_0 = \phi^i(x, y, 0)_{i=1}^N$ for N different phases, the goal is to predict the evolved fields $\mathcal{X}_t : \mathbf{\Phi}_t = \phi^i(x, y, t)_{i=1}^N$ of each phase at different moments. Typically, $\mathbf{\Phi}_t \in \mathbb{R}^{N\times H\times W}$ represents the volume-fraction field in a two-dimensional structured grid of size $H \times W$ with N phases.

To this end, we build a model with learnable parameters Θ that infers the desired mapping by capturing the spatiotemporal dependencies of multiphase volume-fraction fields, $\mathcal{F}_\Theta : \mathcal{X}_0 \rightarrow \mathcal{X}_t$, thereby substituting the direct computation of the governing multiphase-flow PDEs. In our framework, the mapping is organized through two sub-models:

1) Reduced-order model $\mathcal{A}$ for dimensionality reduction and reconstruction:

$$\begin{aligned} \mathcal{A}_{\Theta_1} &= \mathcal{E}_\theta + \mathcal{D}_\varphi, \\ \mathcal{Z}_0, \mathcal{Z}_t &= \mathcal{E}_\theta(\mathcal{X}_0, \mathcal{X}_t), \\ \hat{\mathcal{X}}_0, \hat{\mathcal{X}}_t &= \mathcal{D}_\varphi(\mathcal{Z}_0, \mathcal{Z}_t), \end{aligned} \tag{1}$$

where $\mathcal{E}_\theta$ denotes the complete set of operations in sub-model $\mathcal{A}$ that compress and reduce the dimensionality of the initial flow field. $\mathcal{D}_\varphi$ denotes the complete set of operations in sub-model $\mathcal{A}$ that decode the low-dimensional latent representation and expand it back into the full-resolution field. $\mathcal{Z}$ denotes the latent-space representation. Sub-model $\mathcal{A}$ compresses both the original initial field $\mathcal{X}_0$ and predicted field $\mathcal{X}_t$ into latent space, yielding the representations $\mathcal{Z}_0$ and $\mathcal{Z}_t$, respectively.

2) Neural operator model $\mathcal{B}$ for predicted flow field output using latent representation:

$$\hat{\mathcal{Z}}_t = \mathcal{B}_{\Theta_2}(\mathcal{Z}_0, t), \hat{\mathcal{X}}_t = \mathcal{D}_\varphi(\hat{\mathcal{Z}}_t) \tag{2}$$

Sub-model $\mathcal{B}$ takes the initial latent representation $\mathcal{Z}_0$ together with the time t and predicts the corresponding latent state $\hat{\mathcal{Z}}_t$. The pretrained decoder $\mathcal{D}_\varphi$ then maps $\hat{\mathcal{Z}}_t$ back to the full physical field $\hat{\mathcal{X}}_t$.

Our training objectives of architecture are two folds:

1) Reduced-order training target:

We need to minimize the discrepancy between the original volume-fraction field and its reconstruction produced by sub-model $\mathcal{A}$:

$$\Theta_1^* = \arg\min_{\Theta_1} \mathcal{L}_{\Theta_1}(\mathcal{A}_{\Theta_1}(\mathcal{X}), \mathcal{X}). \tag{3}$$

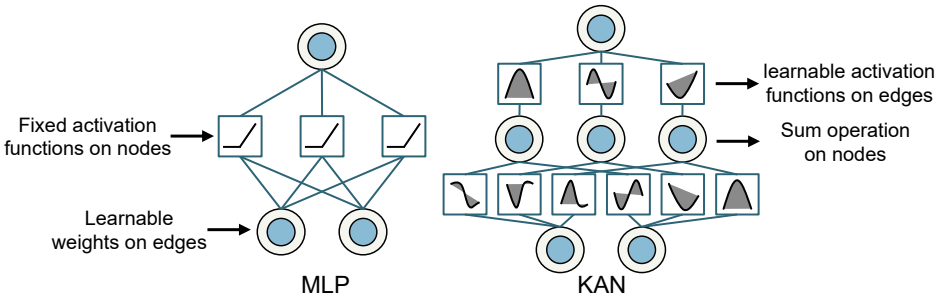

Fig. S1: Architecture comparison between MLP and KAN.

2) Operator training target:

We need to minimize the difference between the latent-space prediction generated by sub-model $\mathcal{B}$ and the true latent representation of the future phase field:

$$\Theta_2^* = \arg\min_{\Theta_2} \mathcal{L}_{\Theta_2}(\mathcal{B}_{\Theta_2}(\mathcal{Z}_0, t), \mathcal{Z}_t), \tag{4}$$

Hence, the optimal internal parameters $\Theta^* = (\Theta_1^*, \Theta_2^*)$ are determined based on the above training processes. $\mathcal{L}_{\Theta_1}$ and $\mathcal{L}_{\Theta_2}$ are corresponding loss functions that are required to be optimized according to the iterative parameter updating.

A.2 GRAMKAN

Kolmogorov-Arnold Networks (KANs) are a recently proposed neural-network architecture designed as a drop-in replacement for conventional multilayer perceptrons (MLPs)[1–3]. As illustrated in Fig. S1, KANs shift the learnable nonlinearities from the neurons themselves to the connecting edges. In a standard MLP, each neuron first forms a weighted sum of its inputs and then passes the result through a fixed activation function such as ReLU. By contrast, KAN neurons perform only the weighted summation; the expressive power arises entirely from smooth, one-dimensional functionstypically implemented with splineslearned on every edge. This edge-wise parameterization enables KANs to achieve higher approximation accuracy with far fewer parameters, making them particularly attractive for scientific-computing tasks that demand precise function representations[4–6].

The original KAN implementation employs B-splines as edge-wise basis functions to approximate the univariate mappings required by the Kolmogorov-Arnold theorem. Although B-splines offer local support and controllable smoothness, their piecewise-polynomial form on a continuous domain entails substantial computational overhead and a large number of trainable parameters. To address these drawbacks we replace B-splines with Gram polynomials, yielding a new variant we term GramKAN.

Gram polynomials—also known as discrete Chebyshev polynomials—possess a simple three-term recurrence and are defined on a finite set of grid points, providing an orthogonal basis with respect to a discrete inner product[7, 8]. This discrete orthogonality not only reduces parameter redundancy but also enhances representation power when modelling inherently discrete data such

as images or token sequences. By exploiting the recursive construction of Gram polynomials, GramKAN achieves comparable or higher approximation accuracy with fewer parameters and lower runtime, making it especially suitable for large-scale scientific and data-centric applications where both precision and efficiency are critical.

Unlike standard continuous-domain polynomials, Gram polynomials are orthogonal on a discrete grid. Selecting m discrete nodes $x(i) = -1 + \frac{2i-1}{m}$ in $[-1, 1]$ and defining discrete inner product:

$$(f, g) = \sum_{i=1}^{m} f(x_i), g(x_i) \tag{5}$$

Under this inner product, Gram polynomials of distinct orders are orthogonal, ensuring that basis functions of different degrees remain independent when capturing discrete data patterns. The Gram polynomials satisfy the following recurrence relation:

$$\begin{aligned} &P(0, m, x) = 1, \quad P(1, m, x) = x \\ &P(n+1, m, x) = xP(n, m, x) - \beta(n, m)P(n-1, m, x), \\ &\beta(n, m) = \frac{(m^2 - n^2)n^2}{m^2(4n^2 - 1)} \end{aligned} \tag{6}$$

The recurrence relation underscores the computational simplicity of Gram transforms, enabling rapid generation of discrete basis functions of any order. To confine inputs to the domain $[-1, 1]$ on which Gram polynomials are defined, we prepend a tanh normalization layer, a step that improves robustness for image and token-sequence data.

Relative to the continuous, locally supported B-spline bases used in the original KAN, Gram polynomials offer distinct advantages. B-splines approximate complex functions by stitching together piecewise polynomials, requiring knot placement and interval bookkeeping; Gram polynomials, by contrast, form a global, orthogonal basis on a discrete grid. Orthogonality decouples the contribution of each degree, reducing parameter interplay during training, while the three-term recurrence eliminates the overhead of managing multiple segments. Most importantly, the discrete nature of Gram bases aligns naturally with pixel- or symbol-level representations, giving GramKAN a cleaner implementation and stronger modelling capacity for inherently discrete inputs. Replacing B-splines with Gram polynomials therefore streamlines the KAN implementation while enhancing its adaptability and representational power for discrete-input tasks.

A.3 KAN-based autoencoder (KAE)

Autoencoders (AEs) are symmetric networks composed of an encoder, which compresses high-dimensional data into a low-dimensional latent code, and a decoder, which reconstructs the original input. Extensive studies show that deep AEs markedly reduce dimensionality while preserving salient information and often outperform linear techniquessuch as principal component analysis—at identical latent sizes. Consequently, AEs have become a standard tool for unsupervised feature learning and dimensionality reduction, providing compact representations for downstream discrim-

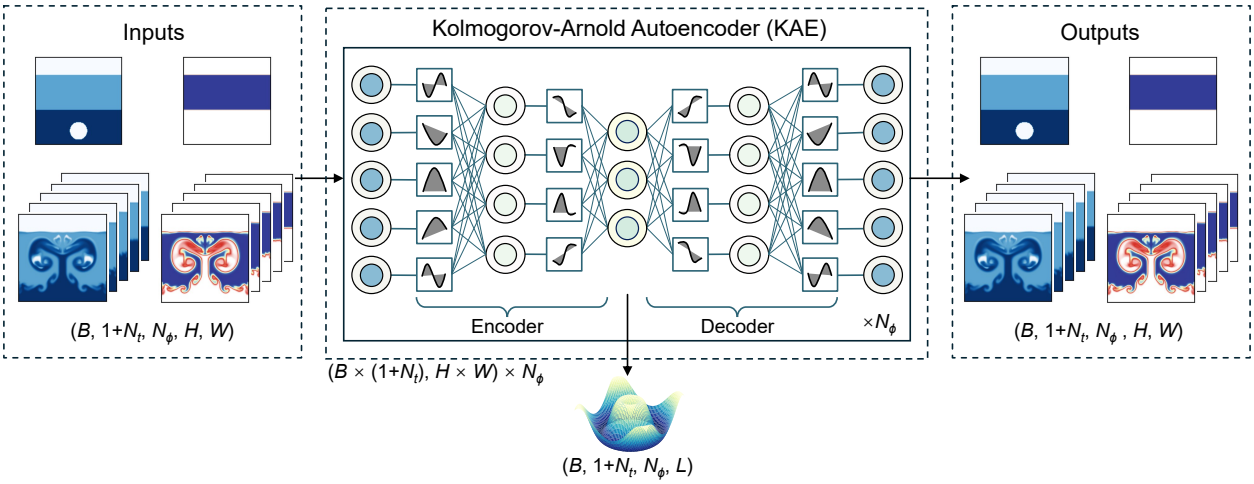

Fig. S2: Proposed KAE architecture.

inative or generative tasks[9–11].

To further enhance compression efficiency and reconstruction fidelity, we couple Kolmogorov-Arnold Networks with an AE, yielding the KAE. Replacing fixed activations in both encoder and decoder with learnable univariate functions on each edge augments nonlinear expressiveness in both compression and reconstruction. In the encoder, stacked KAN layers extract high-order nonlinear features; the decoder mirrors this structure to achieve high-fidelity recovery. Experiments on retrieval, classification and denoising benchmarks show that KAE significantly lowers reconstruction error and improves latent-space separability relative to standard AEs and other KAN variants[3, 12, 13].

KAE is particularly advantageous for multiphase-flow fields, whose data are high-dimensional, strongly nonlinear, feature sharp phase interfaces and are often contaminated by measurement noise. First, the encoder's flexible basis functions (e.g., polynomials or splines) adaptively capture intricate nonlinear patterns, faithfully encoding complex interface geometries and fine-scale vortical textures into a compact latent representation. Second, the decoder's powerful nonlinear mapping not only reconstructs global structures but also restores subtle local details and cross-scale correlations, surpassing linear or shallow nonlinear decoders in fidelity. Finally, by incorporating training tricks such as noise-augmented learning or appropriate regularization, KAE explicitly separates structured physical modes from random noise within the latent manifold, yielding robust denoising even under severe perturbations while preserving the underlying physics.

Fig. S2 shows the KAE architecture and its data pipeline. The model ingests a five-dimensional tensor comprising the volume-fraction field at the initial time t_0 together with N_t subsequent snapshots, yielding an input of shape $(1 + N_t, N_\phi, H, W)$, where N_ϕ is the number of phases and $H \times W$ denotes the spatial resolution.

Assuming a batch of $B = |\mathcal{B}_{\mathrm{KAE}}|$ volume-fraction fields, the input data can be represented as a tensor $\mathcal{B}_{\mathrm{KAE}} \in \mathbb{R}^{B\times(1+N_i)\times N_\phi \times H\times W}$. Since the encoder in the KAE model is designed to process two-dimensional data with an explicit batch dimension, the input tensor must be reshaped. First,

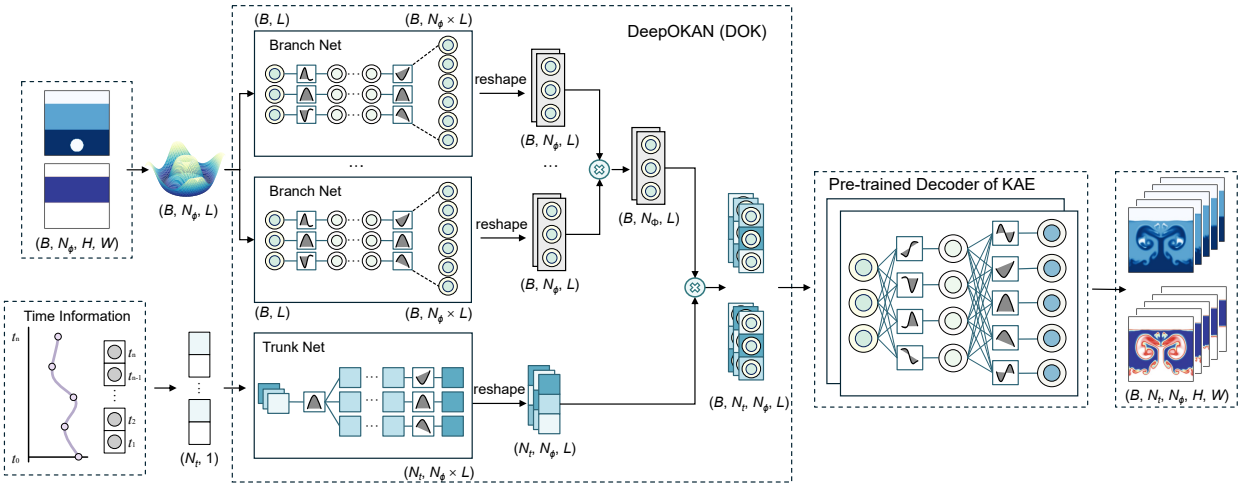

Fig. S3: Proposed DeepOKAN architecture.

the spatial dimensions H and W are merged into a single feature dimension, resulting in $(H \times W)$. Next, according to our architectural design, two reshaping strategies are available. One option is to flatten the batch size B, the temporal dimension $(1 + N_t)$, and the number of phases N_ϕ into a single sample axis, yielding a tensor of shape $(B \times (1 + N_t) \times N_\phi, H \times W)$, which is then processed by a shared KAE for compression and reconstruction. Alternatively, one may flatten only the batch and time dimensions to form $(B \times (1 + N_t), H \times W) \times N_\phi$, and assign an independent KAE to each phase.

The core purpose of this reshaping operation is to treat each sample, each time step, and each phase component as an independent input unit, allowing the encoder to simultaneously process spatiotemporal data from different phases. The reshaped tensor is then passed into the encoder, which compresses the high-dimensional spatial features $(H \times W)$ into a low-dimensional latent representation of length L, where $L \ll H \times W$. Finally, to recover the original batch, time, and phase structure, the encoder output is reshaped into a four-dimensional tensor of shape $(B, 1+N_t, N_\phi, L)$, which compactly encodes the spatial structures of the input time series in latent space.

A.4 Deep Operator Kolmogorov-Arnold Network (DeepOKAN)

Deep Operator Network (DeepONet) is a neural network architecture designed to learn mappings from input functions to output fields[14]. It consists of two subnetworks: a branch net, which encodes the input functions or parameters, and a trunk net, which processes independent variables such as spatial coordinates or time. The outputs of these two subnetworks are fused to predict the target function. Recent studies have demonstrated the capability of DeepONet to approximate complex operator mappings with high accuracy[15–19].

To further enhance its ability to capture highly nonlinear operator relationships, we propose DeepOKAN, a variant of DeepONet in which all standard MLP layers in both branch and trunk nets are replaced with Kolmogorov-Arnold Network (KAN) layers. This substitution increases

the expressive power and nonlinear flexibility of the model. Recent works have also explored the integration of KAN into neural operator frameworks, using Gaussian radial basis functions or B-splines as the basis in KAN layers to better approximate the intricate mappings between input and output fields (see e.g., DeepOKAN).

Applied to the prediction of complex multiphase dynamics, DeepOKAN offers several notable advantages. In the branch net, the adaptable basis functions of KAN can effectively extract nonlinear correlations among latent features of different phases. This enables the network to learn how initial multiphase configurations influence interface evolution and interphase mixing more thoroughly than conventional branch architectures. In the trunk net, KAN's strong nonlinear approximation capacity enables accurate modeling of the latent feature evolution over time, capturing both macroscopic trends and fine-scale local variations—such as interface propagation, vortex generation, and dissipation.

The final output is formed by pointwise multiplication of branch and trunk features, enabling a tightly coupled representation of initial conditions and temporal dynamics. Furthermore, since all predictions are performed in the latent space learned by the preceding KAE module—which already separates physical structure from random noise—DeepOKAN benefits from strong robustness. Even under heavy noise contamination in the initial inputs, the compressed representations and temporal evolution predicted by DeepOKAN yield reconstructions that faithfully reflect the true flow field evolution while suppressing noise amplification.

Fig. S3 illustrates the schematic architecture and data processing workflow of the DeepOKAN model. The input to DeepOKAN is the low-dimensional latent representation obtained from the KAE encoder, corresponding to the multiphase volume fraction fields at the initial time step $t = 0$.

Assuming a batch of $B = |\mathcal{B}_{\mathrm{DOK}}|$ samples, with N_ϕ phases and a latent dimension of L per phase, the entire input to the branch net can be represented as $\mathcal{B}_{\mathrm{DOK}} \in \mathbb{R}^{B\times N_\phi \times L}$. To accommodate the multiphase input structure, we employ N_ϕ parallel sub-branch nets, each responsible for processing the latent vector of a specific phase. Each sub-branch net consists of multiple KAN layers with uniform width, extracting high-order nonlinear features from the input. The final layer of each sub-branch net expands the output to a vector of length $N_\phi \times L$. The outputs from the N_ϕ sub-branch nets are then element-wise multiplied, producing a fused vector of shape $(B, N_\phi \times L)$, which is subsequently reshaped into a latent tensor of shape (B, N_ϕ, L), representing the initial latent state.

On the trunk side, the network receives time inputs of shape, $(N_t, 1)$, where Nt denotes the number of prediction time steps. These temporal inputs are processed through a series of KAN layers, producing outputs of size $N_\phi \times L$, which are reshaped to form a tensor of shape (N_t, N_ϕ, L). Next, the outputs from the branch net (B, N_ϕ, L), and trunk net (N_t, N_ϕ, L) are combined via element-wise multiplication, yielding the predicted latent features with shape $(B_{,t}, N_\phi, L)$. These are then passed through the KAE decoder to reconstruct the full multiphase flow fields in physical space, resulting in outputs of shape (B, Nt, N_ϕ, H, W).

The DeepOKAN model is trained by minimizing the discrepancy between the predicted and ground-truth latent features, thereby ensuring accurate learning of the latent-space evolution dy-

namics of multiphase volume fraction fields.

B Dataset

The dataset encompasses snapshots of the temporal evolution for three representative two-phase flows and one three-phase flow case:

(a) Rising bubble in quiescent water (Bubble rise);

(b) Settling of solid particles in liquid (Particle deposition);

(c) Gas-driven fluidization of solid particles (Fluidized bed);

(d) Gas-liquid-solid multiphase system with bubble-particle interaction (Bubble-particle coupling problem).

As illustrated in Fig. S3, each subfigure presents the computational domain configuration, initial control parameters, and representative transient structures of the volume fraction fields for each case. These cases cover typical interactions in gas-liquid, liquid-solid, gas-solid, and gas-liquid-solid multiphase systems. All simulations were performed using the open-source software OpenFOAM. The configuration of each case, along with its initial conditions, numerical methods, and dataset construction details, is described as follows.

Case 1: Bubble rise (Fig. S4-a) simulates the rising of a single air bubble in stagnant water using the Volume of Fluid (VOF) method to accurately capture the gas-liquid interface. The computational domain is a 2D rectangular tank of size 1 m $\times$ 1 m, with an initial water level set to $H_l = 0.66\,\mathrm{m}$. At $t = 0$, a static circular bubble is initialized at the bottom. Independent samples are generated by varying the initial bubble radius $R_b \in [0.08\,\mathrm{m}, 0.12\,\mathrm{m}]$ and the vertical position of the bubble center $H_b \in [0.12\,\mathrm{m}, 0.42\,\mathrm{m}]$. All boundaries are no-slip walls. Under buoyancy and gravity, the bubble deforms and rises, exhibiting characteristic wake vortices (Fig. S4-a). The simulation lasts for 1 s, and the liquid phase volume fraction is saved at fixed time intervals of 0.1 s.

Case 2: Particle deposition (Fig. S4-b) models the sedimentation of solid particles in water using a two-fluid EulerbEuler model. The computational domain is a 0.5 m $\times$ 0.3 m vertical 2D container with all boundaries set as no-slip walls. At the initial time, solid particles are uniformly suspended in the upper half of the water column, while the lower half is clear water, forming a strongly unstable stratification. Driven by gravity, the denser upper layer sinks, interacting with the lower layer through entrainment and mixing, forming characteristic finger-like falling structures and vortex roll-ups (Fig. S4-b). Independent trajectories are generated by varying the initial particle volume fraction $\phi_p \in [0.5, 0.6]$ and the particle-water interface height $H_s \in [0.1\,\mathrm{m}, 0.2\,\mathrm{m}]$. The simulation lasts 2 s, and particle volume fraction fields are recorded every 0.2 s.

Case 3: Fluidized Bed (Fig. S4-c) investigates gas-solid fluidization using a two-fluid Euler-Euler approach. The computational domain is a vertical 2D container of size 0.5 m $\times$ 0.4 m. A gas inlet (velocity $u_{\mathrm{air}} = 0.1$ m/s) is located at the bottom, a pressure outlet at the top, and the sidewalls are no-slip boundaries. Initially, solid particles are uniformly packed at the bottom, and

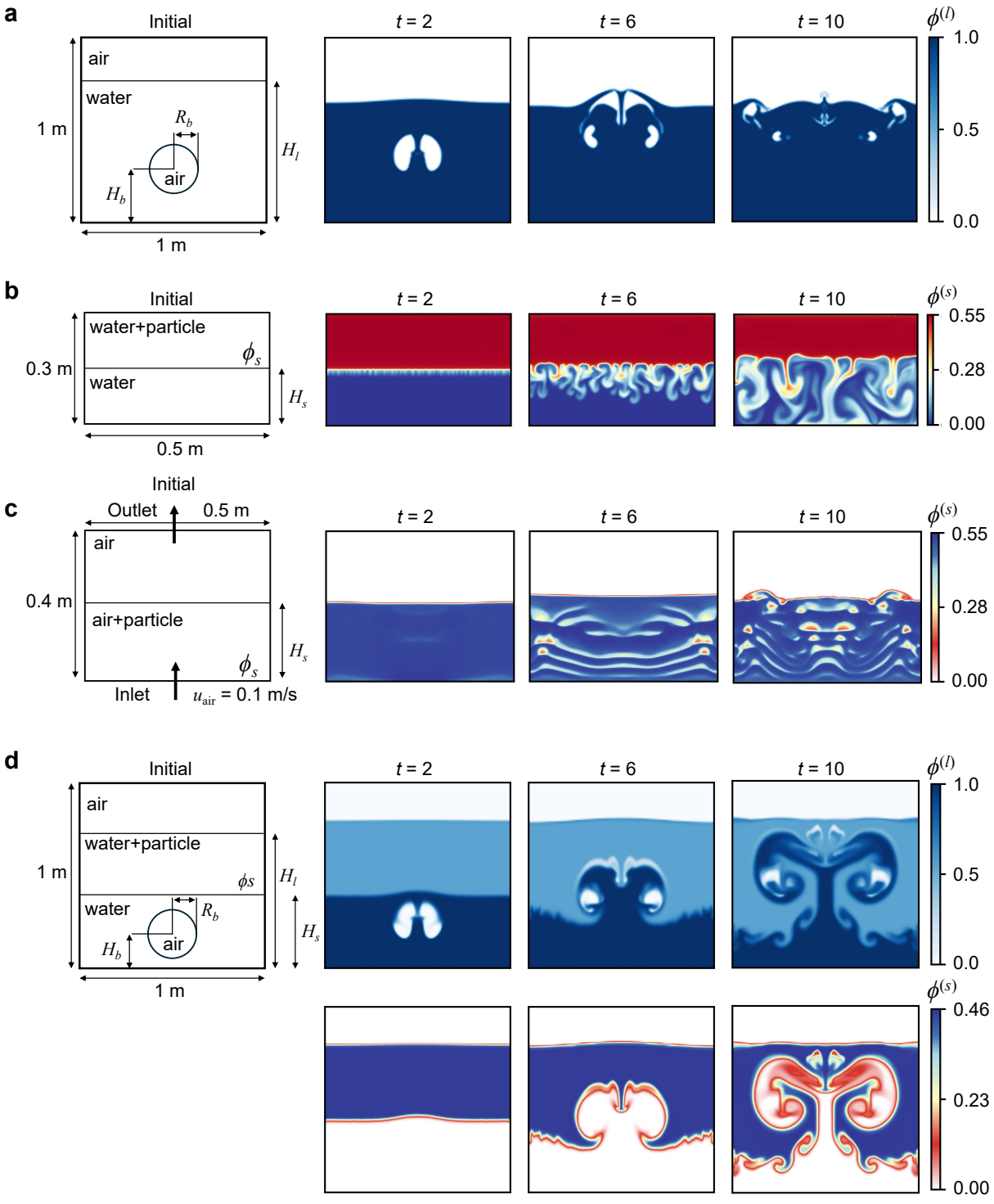

Fig. S4: Multiphase flow cases setting.

the gas phase shares the same velocity as the inlet. This initialization eliminates spurious startup sedimentation and accelerates the transition into the fluidization regime. Once the gas velocity exceeds the minimum fluidization velocity, rising gas bubbles and voids emerge within the bed,

triggering vigorous particle motion (Fig. S4-c). Simulations run for 1 s, and snapshots of the particle volume fraction are saved every 0.1 s. By varying the initial particle volume fraction $\phi_s \in [0.5, 0.6]$ and bed height $H_s \in [0.1\,\text{m}, 0.3\,\text{m}]$, diverse gas-solid fluidization behaviors are captured.

Case 4: Three phases coupling problem (Fig. S4-d) simulates a **gas-liquid-solid three-phase system** using a multiphase Euler-Euler model. The domain is a closed 1 m × 1 m 2D container. Initially, the lower region is filled with water ($H_l = 0.8\,\text{m}$), and solid particles are uniformly suspended in the upper layer, with the bottom interface of the particle layer located at $H_s = 0.4\,\text{m}$. A static air bubble is initialized underwater as a perturbation. Once released, the bubble rises, penetrating the particle layer and inducing coupled motion among all three phases (Fig. S4-d). The simulation duration is 1 s, with liquid (water) and solid (particle) volume fraction snapshots saved every 0.1 s. Independent cases are generated by varying the initial bubble radius $R_b \in [0.08\,\text{m}, 0.12\,\text{m}]$ and particle volume fraction $\phi_s \in [0.4, 0.5]$, enabling broad sampling of three-phase interaction dynamics.

All simulations were performed on a uniform 2D grid with a fixed spatial resolution of 128 × 128. For each case, 1,000 independent samples were generated under randomized combinations of initial parameters. Each sample contains the volume fraction field at the initial time step and 10 subsequent snapshots at uniform time intervals, resulting in 11 frames per sample. Due to the volume-fraction summation constraint $\sum_{i=1}^{N} \phi_i = 1$, only N-1 phase-fraction fields are stored for each sample. Specifically, two-phase cases store only one phase field, while three-phase cases retain two selected phase fields (e.g., liquid and solid). As a result, each snapshot has a spatial dimension of $(N-1) \times 128 \times 128$, where N is the number of phases. The dataset is randomly split into training and testing sets with a ratio of 9:1, and used for model training and evaluation. All simulations adopt standard physical parameters: air as the gas phase, water as the liquid phase, and incompressible solids with a density of approximately 2500 kg/m^3 for the particulate phase.

C Architecture comparison for autoencoders and operators

C.1 Controlled ablation studies

To elucidate the functional roles of distinct architectural components, we conducted a series of *controlled ablation studies*here referring to systematic variations of individual architectural hyper-parameters rather than the removal of entire modules on the KAE and DOK, varying the width of KAN layers (w), the polynomial order of Gram expansions (q), and the network depth (l).

In KAE, the encoder and decoder adopt a symmetric design. Here, w denotes the width of the encoder's input layer (first layer) or the decoder's output layer (last layer), which directly handles raw inputs or reconstructed outputs and thus critically governs compression and reconstruction fidelity. The depth l specifies the number of layers in the encoder/decoder, with widths decreasing geometrically along the depth. The order q is uniform across all KAN layers in the KAE.

For DOK, ablations are applied solely to the branch network operating on KAE-compressed features, while the trunk network remains fixed. In this context, w is the uniform width of all hidden

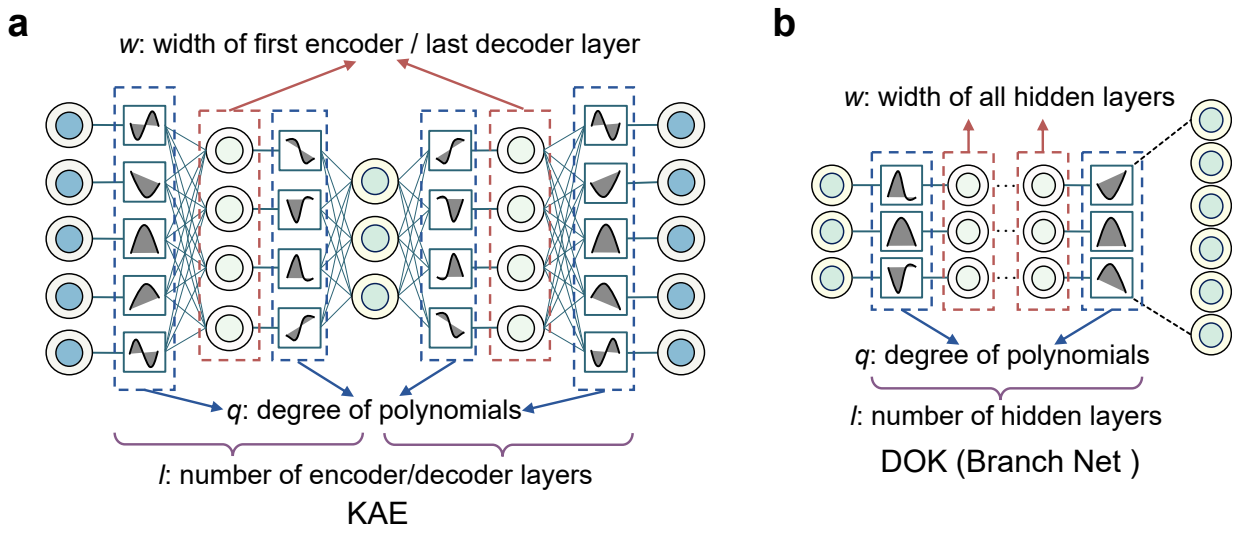

Fig. S5: Controlled ablation settings for KAE and DOK.

Table S1: Baseline models for KAE and DOK

Model	Input Shape	Layers	Degree
KAE	128×128	1008-144	3
DOK-branch	144	288×6	3
DOK-trunk	$(10, 1)$	96×6	3

layers except the output layer, l is the number of hidden layers, and q is the Gram polynomial order of the branch network's KAN layers.

Across all configurations, the base hidden-layer width is fixed at 144. w is expressed as a multiple of this base, i.e., actual width $= 144 \times w$. The parameter search ranges are:

KAE: $q \in \{2, 3, \ldots, 8, 9\}, w \in \{3, 5, \ldots, 17, 19\}, l \in \{1, 2, \ldots, 8, 9\}$

DOK: $q \in \{2, 3, \ldots, 5, 6\}, w \in \{1, 2, \ldots, 8, 9\}, l \in \{2, 3, \ldots, 9, 10\}$

Baseline configurations (Table S1) are defined as $l = 2, w = 7, q = 3$ for the KAE, and $l = 6, w = 2, q = 3$, for the DOK branch network, with the DOK trunk network fixed at $l = 6, w = 2/3, q = 3$. Due to the symmetry of the KAE, Table 1 reports only encoder layer widths, while in the DOK the reported layers exclude the final output layer, whose width is $N_\phi \times 144$, where N_ϕ is the number of phases input to the DOK ($N_\phi = 1$ for two-phase cases and $N_\phi = 2$ for three-phase case).

C.2 Detailed design of architecture

For model comparisons, we should note that the KAN is an MLP-derived architecture. In addition to the conventional MLP-KAN comparison, we further introduce convolution-based variants of both MLP and KAN to isolate the effect of incorporating KAN layers[20, 21]. Fig. S6 illustrates the base-layer designs for each model. The Base Linear layer uses SiLU as the activation function,

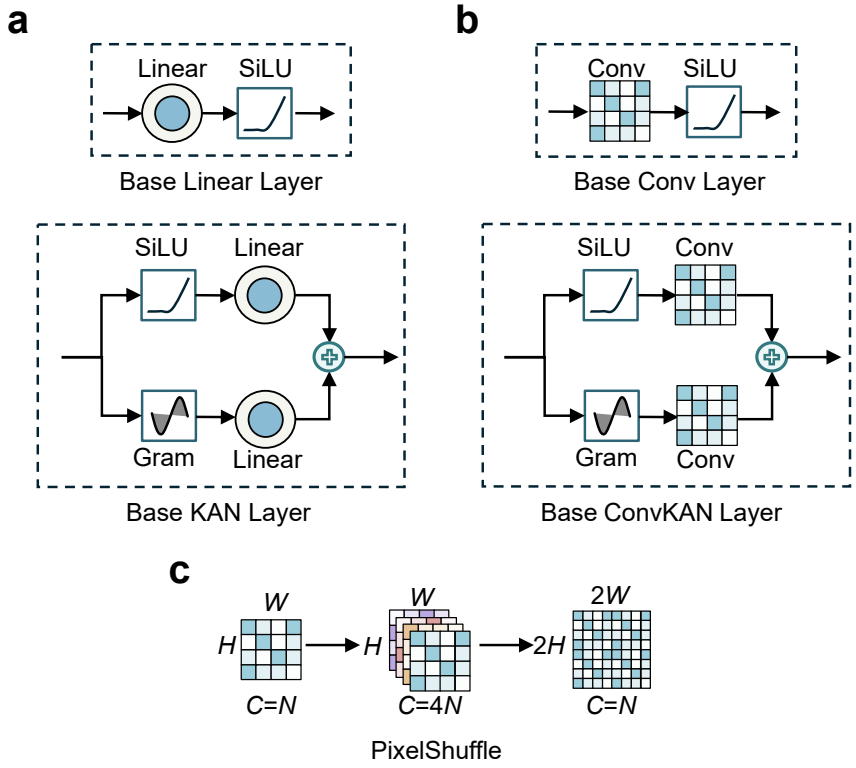

Fig. S6: Base-layer design for different architectures.

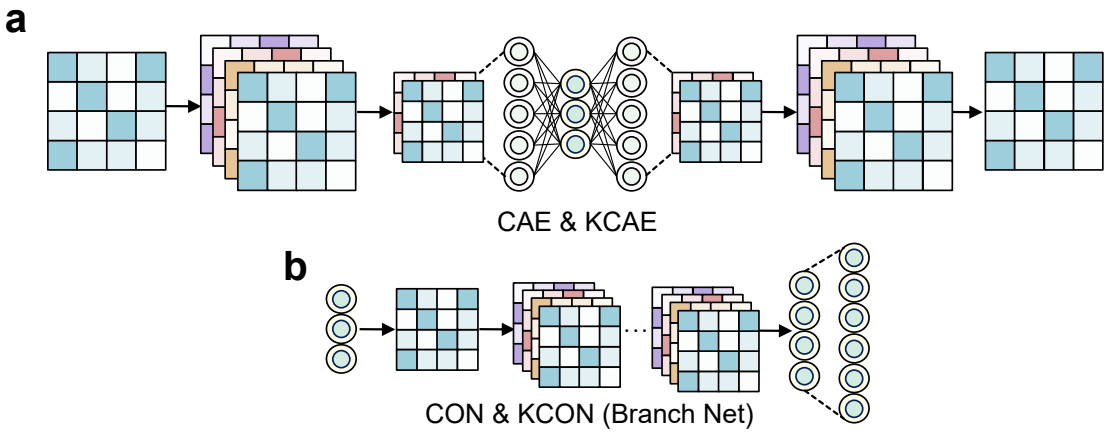

Fig. S7: Architectures of CAE, KCAE, CON, and KCON models based on Base Conv and Base ConvKAN layers.

while the Base KAN layer couples SiLU basis functions with learnable Gram polynomials. In the Base Conv and Base ConvKAN layers, the Linear modules in the Base Linear and Base KAN designs are replaced with 3×3 convolutional modules.

For the dimensionality-reduction and reconstruction task, we designed four autoencoder variants: the linear-layer-based AE and KAE, and the convolution-layer-based CAE and KCAE. The AE and KAE share a similar architecture, each comprising multiple Base Linear or Base KAN

Table S2: Architectural parameters and performance of autoencoder variants. In "Layers", values for AE and KAE correspond to neuron counts per layer, whereas for CAE and KCAE, all but the last value represent convolutional channel numbers.

Model	Type	Layers	Degrees	Parameters (M)	Flops (M)	Epoch Time (s)
AE-H	$(l = 4, w = 33)$	4752-1486-464-144	–	171.41	171.42	2.30
AE-L	$(l = 2, w = 7)$	1008-144	–	33.34	33.33	1.62
KAE-Base	$(l = 2, w = 7, q = 3)$	1008-144	3	166.68	166.61	2.38
KAE-Lw	$(l = 2, w = 3.5, q = 3)$	504-144	3	83.37	83.31	1.65
KAE-Hw	$(l = 2, w = 10.5, q = 3)$	1512-144	3	250.00	250.00	3.21
KAE-Lq	$(l = 2, w = 7, q = 2)$	1008-144	2	133.34	133.29	1.94
KAE-Hq	$(l = 2, w = 7, q = 4)$	1008-144	4	200.02	199.93	2.78
CAE-H	$(l = 4, c = 80)$	80-40-20-16-10-144 (linear)	–	0.38	173.96	2.10
CAE-L	$(l = 4, c = 26)$	26-20-16-10-144 (linear)	–	0.23	33.61	2.01
KCAE	$(l = 4, c = 26, q = 3)$	26-20-16-10-144 (linear)	3	1.16	165.63	3.06

layers (Fig. S2). In contrast, the CAE and KCAE employ, in the encoder, a series of Base Conv or Base ConvKAN layers with a stride of 2 to progressively downsample the input. To ensure a consistent latent-space dimensionality across all models, the downsampled two-dimensional feature maps are flattened into one-dimensional vectors before being processed by a Base Linear or Base KAN layer. The decoder mirrors the encoder's structure and restores the processed data to its original spatial resolution via upsampling. We evaluated three upsampling strategies: bilinear interpolation (followed by a convolution layer for feature refinement), transposed convolution, and PixelShuffle. Bilinear interpolation offers simplicity, while transposed convolution can introduce distortions due to its aggressive upsampling. PixelShuffle, originally developed for super-resolution tasks (Fig. S6-c), first increases the channel dimension by a factor of r^2 via convolution, then rearranges the channels to enlarge the spatial dimensions (H and W) by a factor of r while reducing channels by r^2. Compared with bilinear interpolation, PixelShuffle achieves higher representational capacity with the same number of parameters and improved computational efficiency[22]. We therefore adopted PixelShuffle with an upsampling factor $r = 2$ to match the stride-2 downsampling in the encoder.

For the latent-space prediction task, we likewise developed four operator models: the linear-layer-based DON and DOK, and the convolution-layer-based CON and KCON. All models use a trunk network that accepts scalar time inputs. Because such inputs cannot be directly accommodated by a convolutional architecture—and to maintain comparability across models—the trunk nets in all four variants comprise stacked Base Linear or Base KAN layers of uniform width. The branch net, in contrast, processes one-dimensional latent vectors. For CON and KCON, these vectors are reshaped into two-dimensional feature maps, enabling feature extraction via multiple Base Conv or Base ConvKAN layers with identical channel counts and a stride of 1. The extracted features are then flattened, passed through a Base Linear or Base KAN layer for mapping, and output with the same dimensionality as the DON/DOK branch nets.

Tables S2 and S3 summarize the architectural parameters and performance metrics of the

Table S3: Architectural parameters of neural operator variants. In "Branch/Trunk Layers", values for DON and DOK denote (neurons × layers), whereas for CON and KCON they denote (convolutional channels × layers).

Model	Type	Branch Layers	Trunk Layers	Degrees	Parameters (M)	Flops (M)	Epoch Time (s)
DON-H	$(l = 6, w = 5.5)$	792×6	288×6	–	4.93	11.67	1.11
DON-L	$(l = 6, w = 2.5)$	288×6	96×6	–	0.95	2.38	0.98
DOK-Base	$(l = 6, w = 2, q = 3)$	288×6	96×6	3	4.74	11.80	2.56
DOK-Lw	$(l = 6, w = 1, q = 3)$	144×6	96×6	3	2.24	8.31	1.65
DOK-Hw	$(l = 6, w = 4, q = 3)$	576×6	96×6	3	13.23	18.89	3.20
DOK-Lq	$(l = 6, w = 2, q = 2)$	288×6	96×6	2	3.79	9.44	1.94
DOK-Hq	$(l = 6, w = 2, q = 4)$	288×6	96×6	4	5.68	14.15	3.04
CON-H	$(l = 6, c = 16)$	16×6	288×6	–	3.42	11.85	2.94
CON-L	$(l = 6, c = 4)$	4×6	96×6	–	0.83	2.36	0.95
KCON	$(l = 6, c = 4)$	4×6	96×6	3	4.11	11.72	2.83

autoencoder and operator models, respectively. In the operator models, all inputs are latent-space representations generated by the KAE-Base model.

C.3 Visualization of prediction for different architectures

To more clearly illustrate the predictive capabilities of different models, we selected six autoencoder variants and six neural operator variants (twelve models in total) and plotted the temporal evolution of phase volume fraction errors across different time steps. Because the inter-model errors are small, we employed a logarithmic colour scale to enhance visual contrast. To avoid singularities in regions with zero error, we introduced a coefficient SLinear, and defined the linear mapping range as $[-S_{\text{Linear}} \times \Delta\text{Max}, S_{\text{Linear}} \times \Delta\text{Max}]$, where ΔMax is the maximum absolute error. Additionally, small markers (mini tricks) were placed just above and below the zero-value region of the colour bar to clearly delineate this linear zone. The visual comparison results are shown in Fig. S8- S11. A holistic analysis of the results shows that the overall error distribution patterns are consistent with those discussed in the main text. Incorporating KAN or ConvKAN layers markedly improves the dimensionality reduction and reconstruction accuracy of the autoencoder models. In contrast, the corresponding accuracy gains in neural operator models are more modest. Notably, in the current comparisons, ConvKAN methods yielded slightly lower accuracy than their conventional convolutional counterparts.

D Noise-tolerance generalization

D.1 Experimental design for noise-robustness evaluation

To evaluate the robustness of the models to input perturbations, we applied zero-mean Gaussian multiplicative noise to all flow-field data. The noise is defined by multiplying the original field values with a random matrix $(1 + \eta)$, where the noise matrix is nonzero only at randomly selected grid points and satisfies $\eta \sim N(0, \sigma)$; at unselected locations $\eta = 0$. The noise intensity

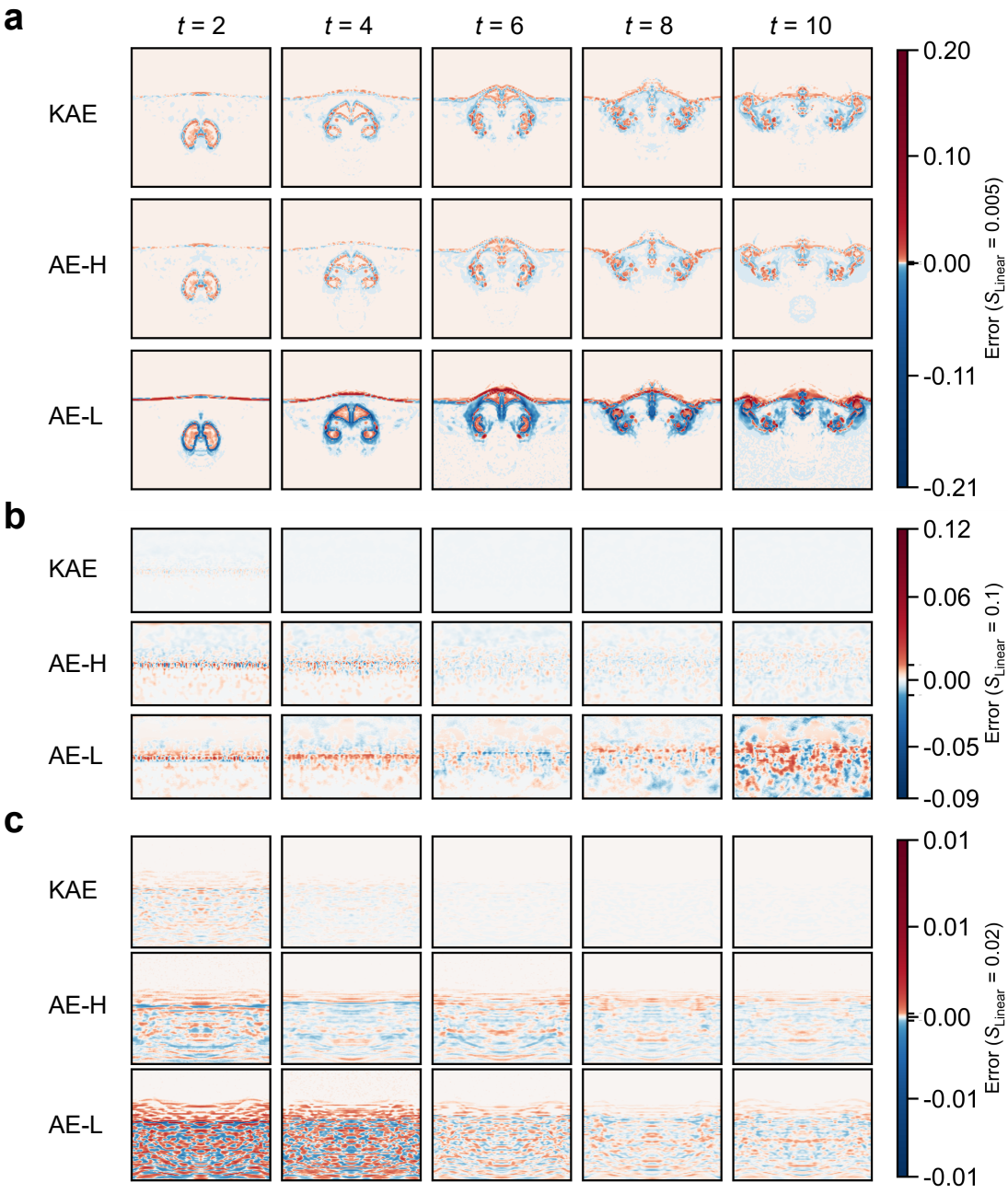

Fig. S8: Prediction errors of KAE, AE-H, and AE-L models across different test cases. **a**, Bubble rise; **b**, Particle deposition; **c**, Fluidized bed.

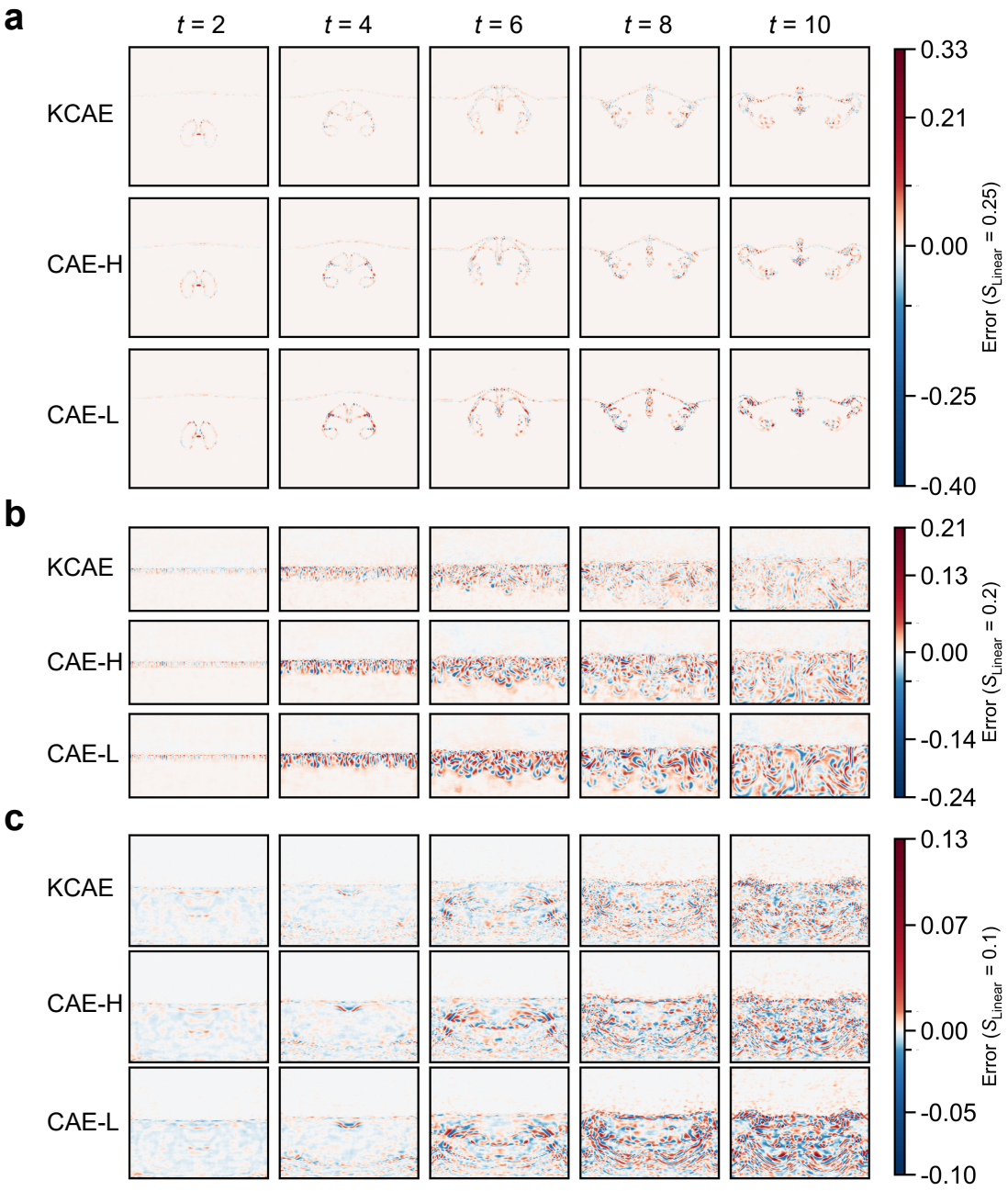

Fig. S9: Prediction errors of KCAE, CAE-H, and CAE-L models across different test cases. **a**, Bubble rise; **b**, Particle deposition; **c**, Fluidized bed.

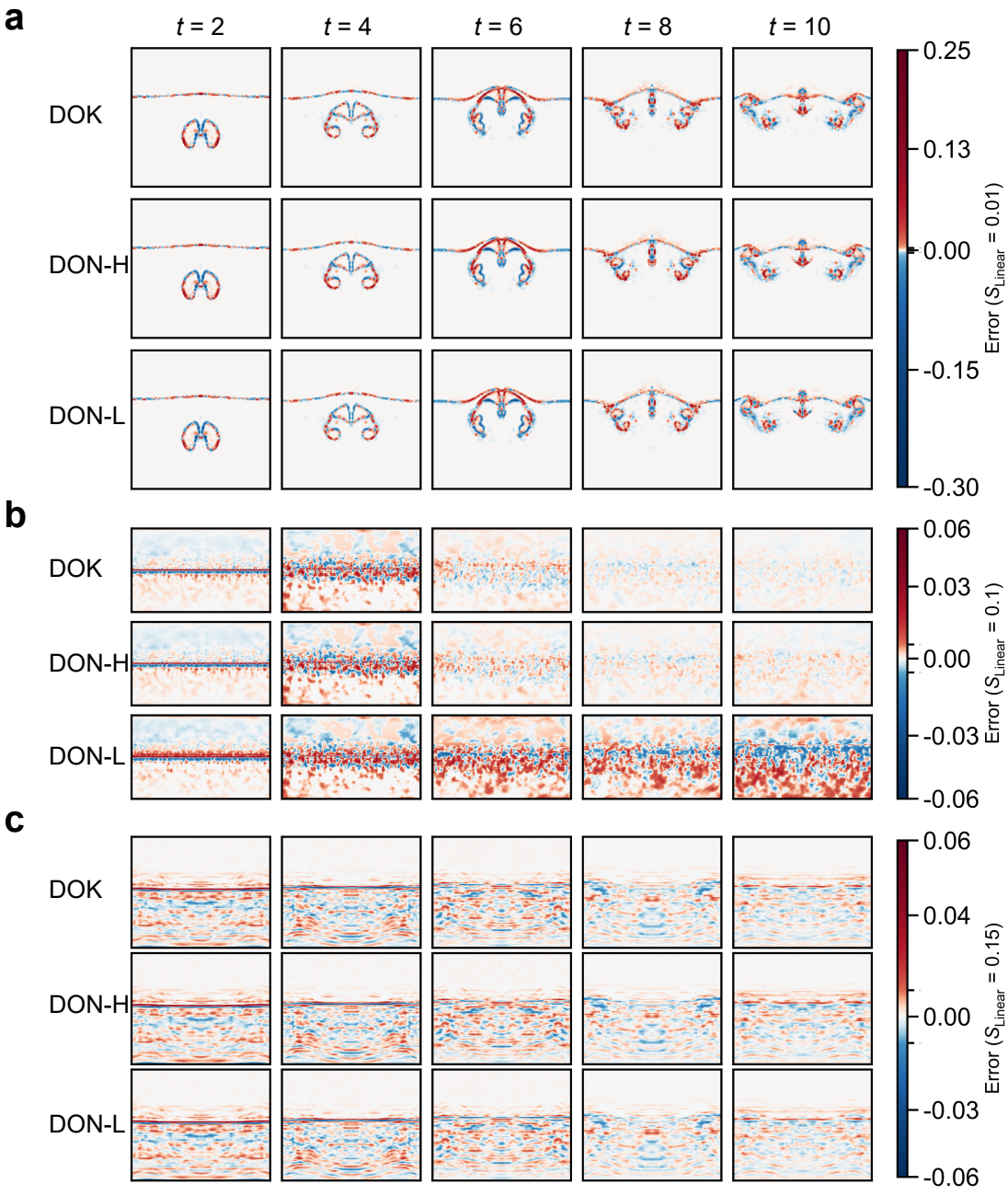

Fig. S10: Prediction errors of DOK, DON-H, and DON-L models across different test cases. **a**, Bubble rise; **b**, Particle deposition; **c**, Fluidized bed.

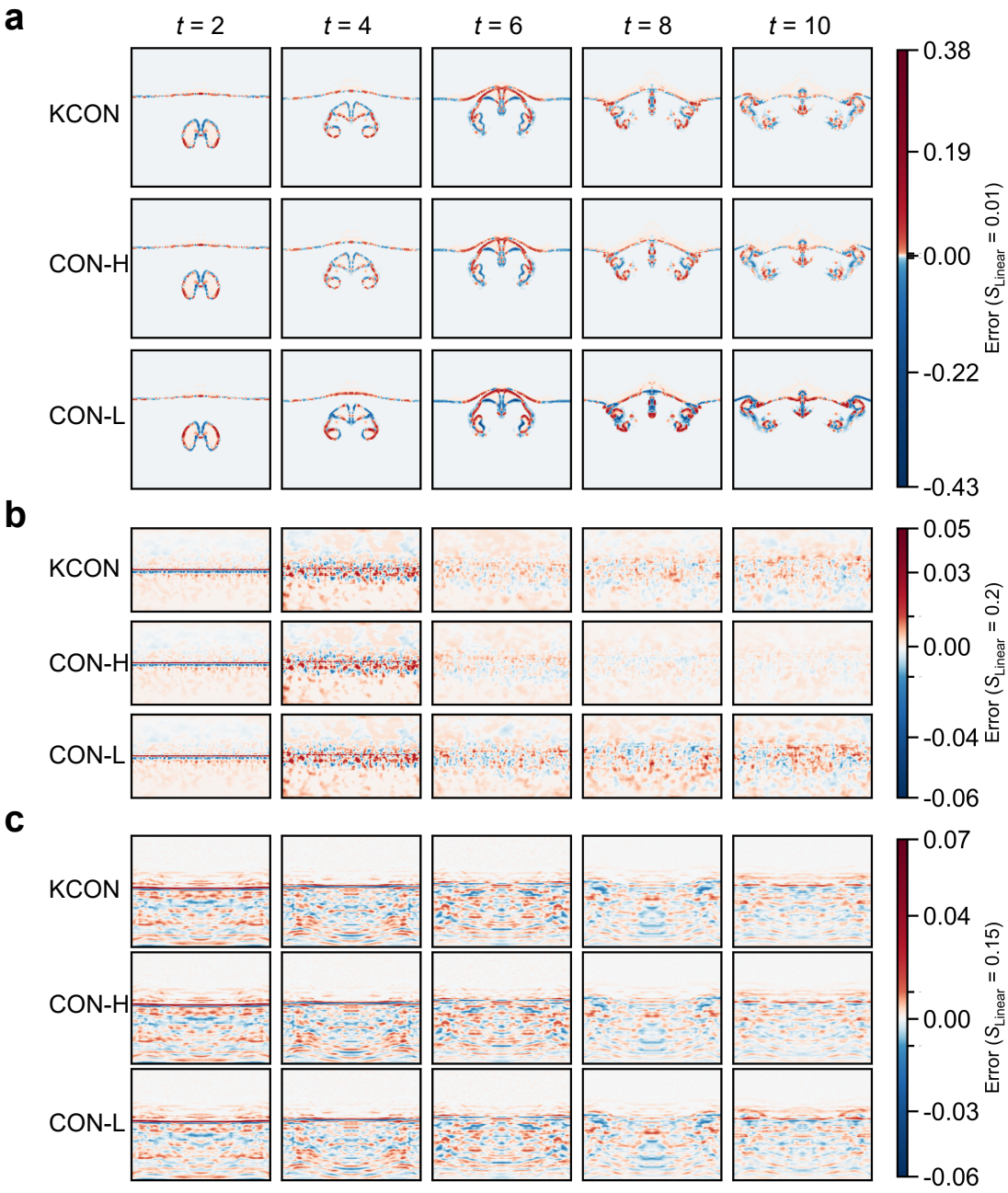

Fig. S11: Prediction errors of KCON, CON-H, and CON-L models across different test cases. **a**, Bubble rise; **b**, Particle deposition; **c**, Fluidized bed.

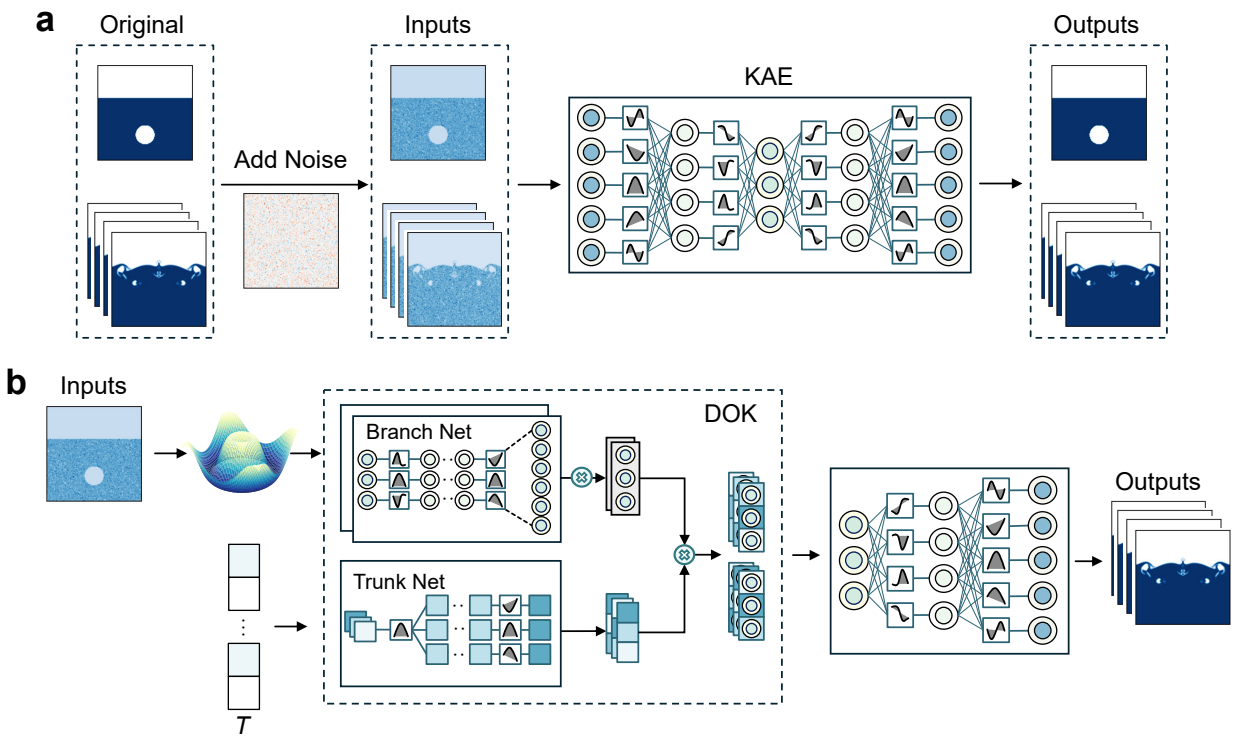

Fig. S12: Schematic diagram of the noise-experiment workflow.

is expressed as a percentage of the original field amplitude. Five intensity levels were considered: $\eta = 5\%, 10\%, 20\%, 30\%, 40\%$. For example, a 10% noise intensity indicates that, at masked positions, η follows a Gaussian distribution with zero mean and standard deviation equal to 0.1 times the original value.

The proportion of perturbed locations (noise ratio) ρ was likewise set to five levels: $\rho = 5\%, 10\%, 20\%, 30\%, 40\%$. For each specific combination of noise intensity σ and noise ratio ρ, we randomly selected the corresponding proportion of grid points within the field to apply the perturbation , leaving all other points unchanged. This yielded 25 distinct noise-evaluation scenarios ($5\sigma \times 5\rho$). The perturbed flow field is expressed as:

$$u_{\text{noisy}}(x, y, t) = u_{\text{orig}}(x, y, t)[1 + \eta(x, y, t)], \tag{7}$$

Here, $\eta(x, y, t)$ takes Gaussian-distributed values at perturbed locations and is zero elsewhere. Notably, this noise-adding procedure is applied to all time frames, including the initial state and all subsequent time steps of the phase volume-fraction fields. This design enables a comprehensive assessment of the model's robustness to input perturbations throughout the entire temporal evolution.

As illustrated in Fig. S12, the experimental workflow proceeds as follows. First, Gaussian noise is added to the flow fields at all time steps in the original dataset (including the initial frame and the subsequent evolution sequence) according to the procedure described above, yielding perturbed

input data for model training and testing. Second, the full noisy time series is fed into the KAE, which is trained to reconstruct the corresponding noise-free flow-field sequence. Through training, the KAE encoder learns to compress high-dimensional noisy fields into low-dimensional latent vectors, while the decoder learns to reconstruct the clean fields from these latent representations. After training, the encoder is capable of extracting robust feature representations from noisy inputs that correspond to the structural patterns of the original flow fields.

In the prediction stage, only the noisy initial field is passed through the trained KAE encoder to obtain its low-dimensional latent representation. This vector encapsulates the essential features of the initial flow while suppressing noise due to the encoder's denoising capability. The initial latent vector is then supplied to the branch network of the DOK model, while the target prediction time t_i is input to the trunk network. For each future time t_i, the trunk network combines the temporal parameter with the initial latent information provided by the branch network to produce the corresponding latent vector ξ_t. This continuous-time modelling in the latent space facilitates smoother temporal evolution and mitigates the influence of noise on predictions at different time steps. Finally, the sequence of latent vectors $\{\xi_t\}$ predicted by the DOK for all specified time steps is passed through the trained KAE decoder to reconstruct the complete flow-field distributions at those times. This procedure is applied identically to each target time in the sequence, thereby enabling multi-step predictions of future flow fields from an initial noisy state.

D.2 Visualization of noise-experiment results

To illustrate the impact of noise injection on model outputs, we visualized results for representative test cases. Figures S13-S15 show the final-frame comparisons for the bubble-rise, particle-deposition, and fluidised-bed cases, respectively, under five noise-level combinations: (5%, 5%), (10%, 10%), ..., (40%, 40%). Each figure presents the KAE and DOK outputs along with the corresponding absolute prediction errors for the given noise condition. In all cases, the first row displays the noisy input flow fields fed to the KAE, serving as a reference for the pre-denoising state.

In the bubble-rise case, increasing noise amplitude and ratio leads to a gradual growth of interface-contour errors in the KAE reconstructions. Nevertheless, even under the $(40\%, 40\%)$ condition, the bubble outline remains clearly recognizable, indicating that the KAE preserves the fidelity of the primary interface features. By comparison, the DOK reconstructions maintain similarly well-defined gas-liquid interfaces across all noise levels, but with slightly larger overall error magnitudes. This suggests that temporal prediction in the latent space smooths part of the high-frequency noise while sacrificing a small amount of local accuracy in single-frame reconstructions. In all cases, the dominant error regions are consistently localized at the bubble diffusion boundary, where the sparse, single-interface structure tends to amplify the effect of strong noise.

In the particle deposition case, increasing noise levels lead to a uniform rise in KAE errors, yet the boundaries of key structures—such as particle-dispersion wakes and recirculation vortices—remain clearly delineated. The error distribution is relatively homogeneous, without pronounced

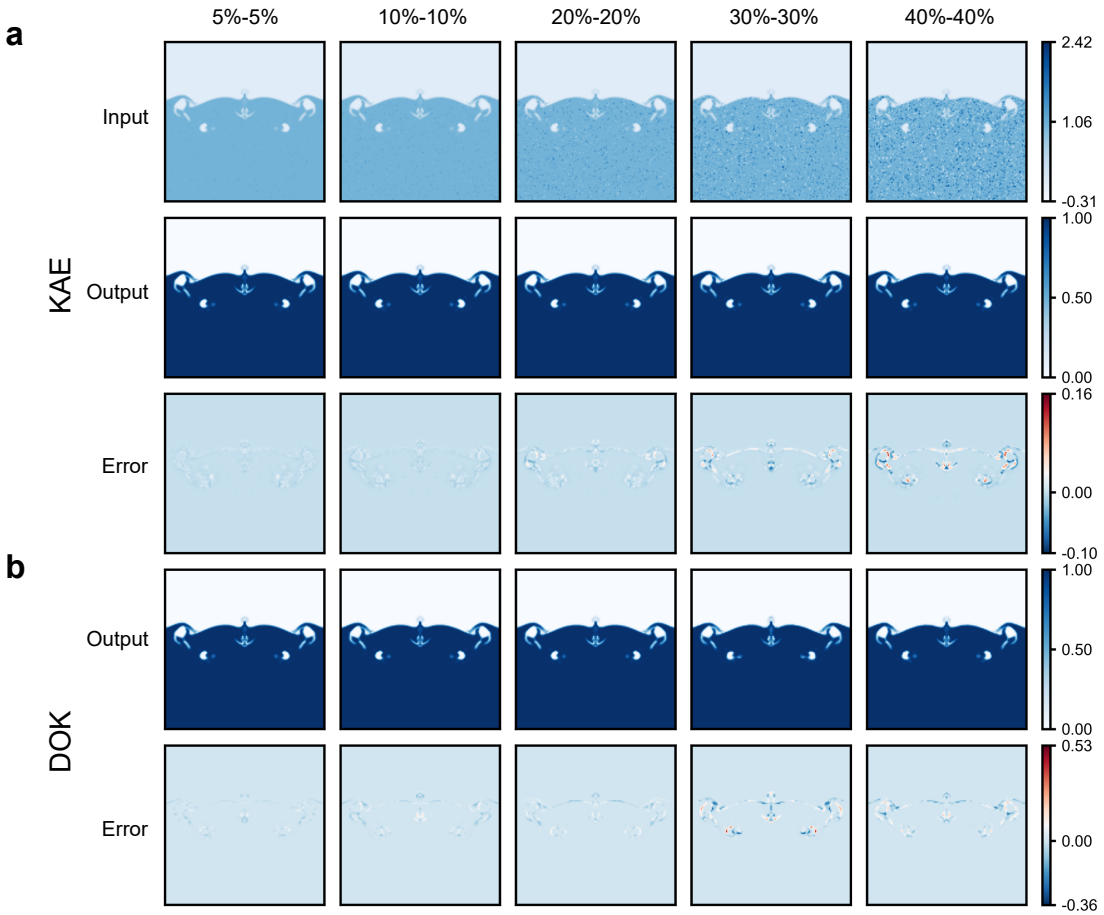

Fig. S13: Predictions of flow field evolutions for the bubble rise case in noise experiment.

localised concentrations. In contrast, across five noise combinations, the DOK reconstructions from multiple independent trials exhibit highly consistent results, with no perceptible change in error distribution or magnitude as noise increases. Compared with the bubble-rise case, the particle-deposition flow field spans a larger spatial domain, displays more complex morphologies, and undergoes more pronounced feature variations. Because such strong and diverse spatial information is preserved with high contrast after encoding, latent-space temporal propagation can more reliably capture critical patterns, thereby rendering DOK comparatively less sensitive to noise in this scenario.

In the fluidized bed case, higher noise levels cause KAE errors to rise uniformly across the domain, yet phase-phase boundaries remain well discernible. The errors form small patch-like distributions along regional boundaries, with relatively larger deviations at the top interface between particles and air. Notably, for DOK, the overall error exhibits a slight "decline" at higher noise levels—both in regions densely populated by solid particles and along bubble-flow channels. This effect may arise because strong noise, once compressed by the encoder, is mapped to latent-space perturbations of smaller amplitude and more dispersed directionality, acting as a mild regularization on the features. As a result, DOK reduces its reliance on high-frequency details and focuses on

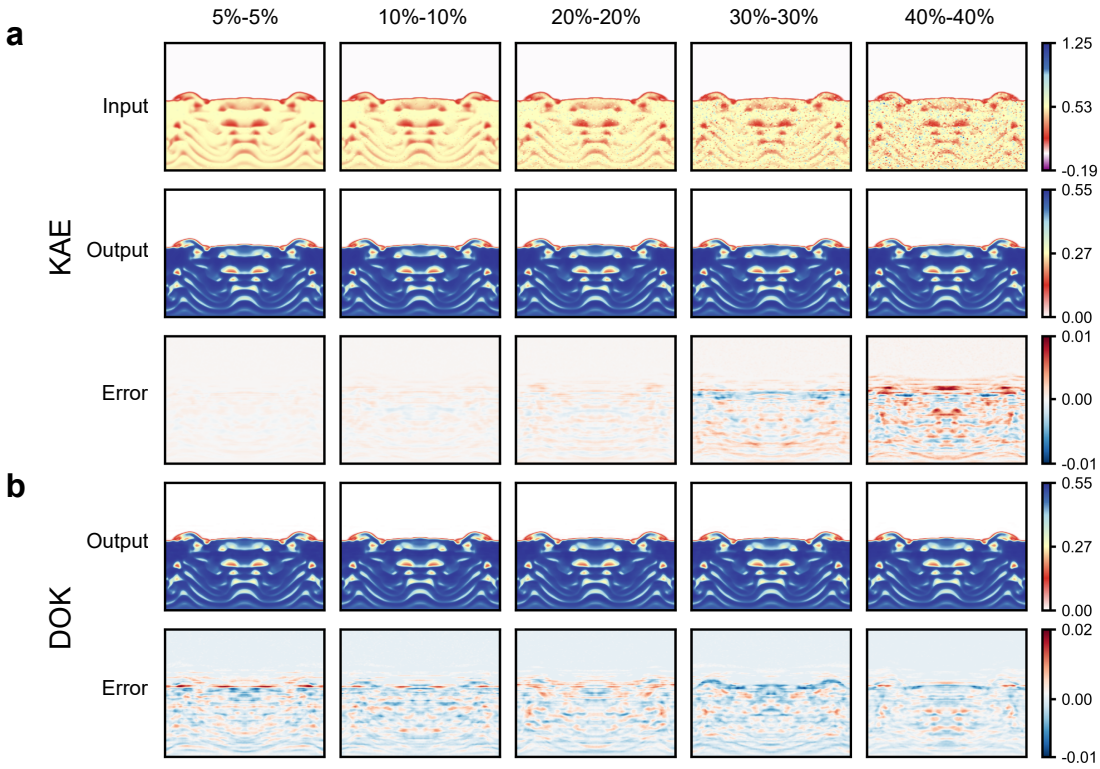

Fig. S14: Predictions of flow field evolutions for the particle deposition case in noise experiment.

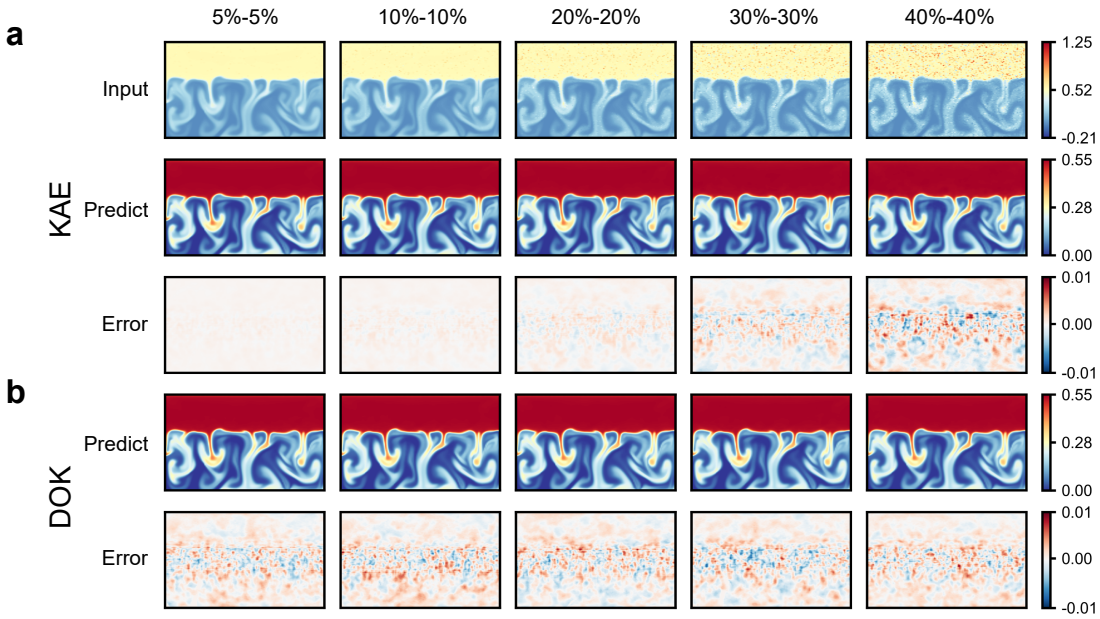

Fig. S15: Predictions of flow field evolutions for the fluidized bed case in noise experiment.

dominant patterns such as bed expansion and bubble-channel formation, thereby achieving more accurate temporal predictions.

Across all cases, the influence of noise on predictive accuracy is inversely related to the spatial-

feature density of the flow field. In interface-sparse scenarios such as bubble rise, noise directly affects the entire region, leading to rapid error accumulation and significant degradation in interface recognition. In contrast, particle deposition and fluidized-bed flows contain abundant, spatially dispersed multiscale structures, causing random perturbations to be "diluted" in the latent space. The denoising capability of KAE and the temporal modelling of DOK together act as a form of implicit regularization, making the overall error less sensitive to noise—and in some high-noise cases, even improving prediction accuracy. This suggests that future model designs could enhance predictive robustness under strong noise by increasing feature redundancy in the flow field and introducing explicit regularisation in the latent space.

E Learning potential for fewer training size and finer temporal resolution

This section presents two experimental designs to evaluate the proposed models under (i) limited training size and (ii) varying numbers of prediction time steps, using the same baseline architecture (Table S1).

In the limited-data scenario, the test set was fixed at 100 samples, while the training set size was varied. The full set of 900 training samples was taken as the 100% baseline, and subsets corresponding to 20%, 30%,..., and 90% of the full data were used for model training and evaluation. For each training ratio, subsets were randomly sampled and the experiments were independently repeated five times with different random seeds to ensure statistical consistency. At each data scale, we evaluated the reconstruction accuracy of the KAE and the prediction accuracy of the DOK (decoded through the KAE), both measured using mean squared error (MSE). Results were compared against the baseline trained on the full dataset to quantify the effect of reduced training data on reconstruction and prediction performance.

The second set of experiments examined model performance under different numbers of prediction time steps. In the baseline configuration, each sample comprised the initial field and ten subsequent flow-field frames (11 frames in total), spanning the evolution from the initial time to a fixed final time T. Keeping the total duration T fixed, we used higher time-resolution simulation data to extend the sequence length to 10, 15, 20, 25, and 30 frames, where more frames correspond to shorter intervals between successive time steps and require the model to predict more intermediate states. For each configuration, the KAE was trained on flow fields from all time frames in that setting (with higher frame counts providing more unsupervised training samples), while the DOK learned to map from the same initial field (branch input) and varying time parameters (trunk input) to the corresponding latent vectors. After training, reconstruction errors for the KAE and prediction errors for the DOK (decoded through the KAE) were computed on the corresponding test sets, both reported as MSE. Each configuration was trained and tested five times with different random seeds to ensure robustness. Comparing error trends across time-step densities enables assessment of how prediction accuracy and stability vary with increasing temporal resolution requirements.

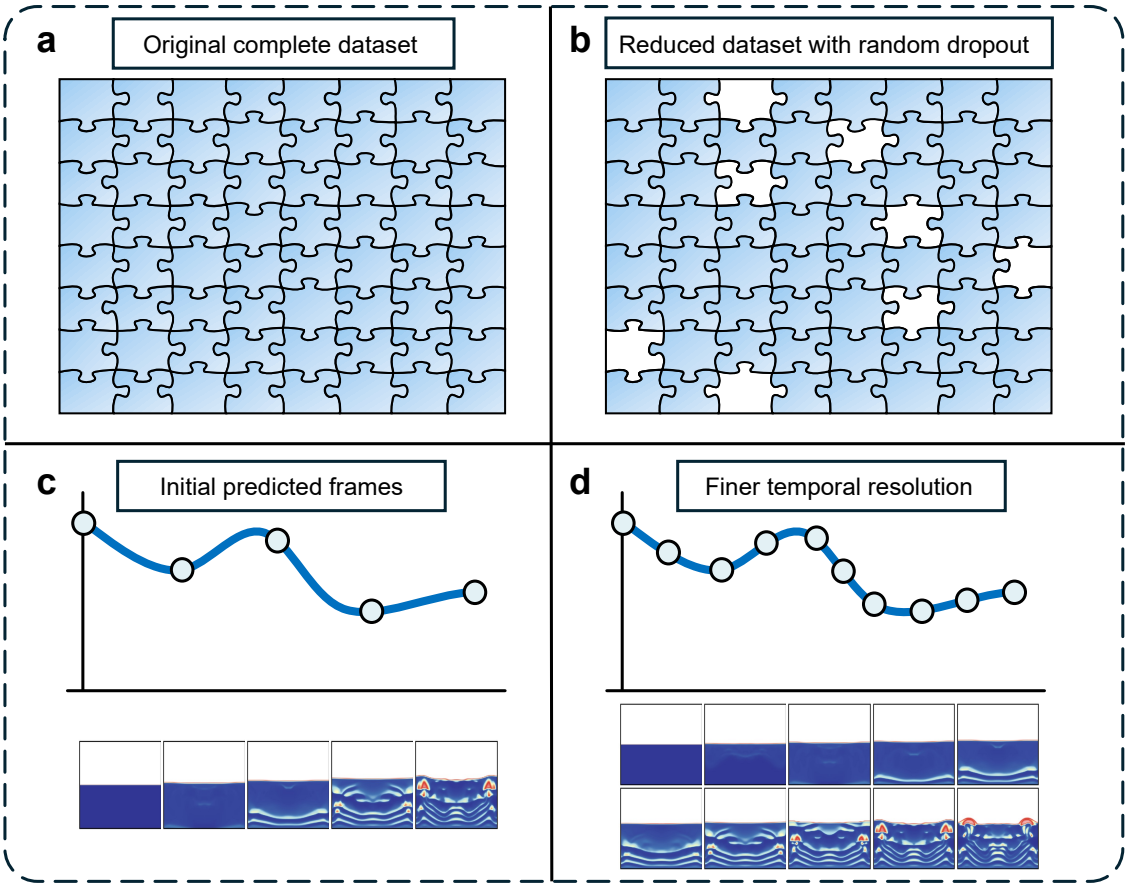

Fig. S16: Experimental designs under limited training size and different time steps.

E.1 Learning potential for fewer training size and finer temporal resolution

As the proportion of training data increases from 20% to 100%, the MSE of both modules decreases monotonically. Across all proportions, the KAE consistently achieves lower errors than the DOK, with the largest gap observed at 100% data availability—consistent with the fact that the DOK's lower error bound is constrained by the reconstruction error of the KAE's latent space. A clear capacity threshold is evident at approximately 40%: for 40% training data, both modules produce stable predictions with well-recovered global structures and fine details; for <40%, errors rise sharply, with notable degradation in flow-field texture and phase alignment. Even at 20-30% data availability, the models can reconstruct the primary outline in all cases, but discrepancies are more pronounced in sensitive regions such as interfaces, shear layers, and single-valued features. In particular, the fluidised-bed case shows the most significant error growth, with numerous high-error spots appearing in the upper air region. These deviations steadily diminish as the training proportion increases to 60-100%. Overall, ensuring at least ∼40% of effective training samples is necessary for reliable predictions, with greater data availability improving KAE reconstruction accuracy and, in turn, enhancing DOK temporal-prediction performance based on its latent space.

Fig. S17: Predictions of flow field evolution under fewer training size tests. **a** Bubble rise; **b** Particle deposition case; **c** Fluidized bed.

E.2 Experimental results for different time step prediction

In the bubble-rise case, KAE errors show a slight decrease as the number of time steps increases; in particle deposition and fluidized bed cases, KAE errors rise slightly with step count, though the magnitude is small. The DOK is more sensitive to temporal densification: for bubble rise, the error increases sharply from 10 to 20 steps but returns to a level comparable to 10 steps at 25-30 steps; in particle deposition, prediction accuracy declines steadily with more steps, as wake and shear-band textures gradually blur while the overall settling profile remains identifiable; in the fluidized bed,

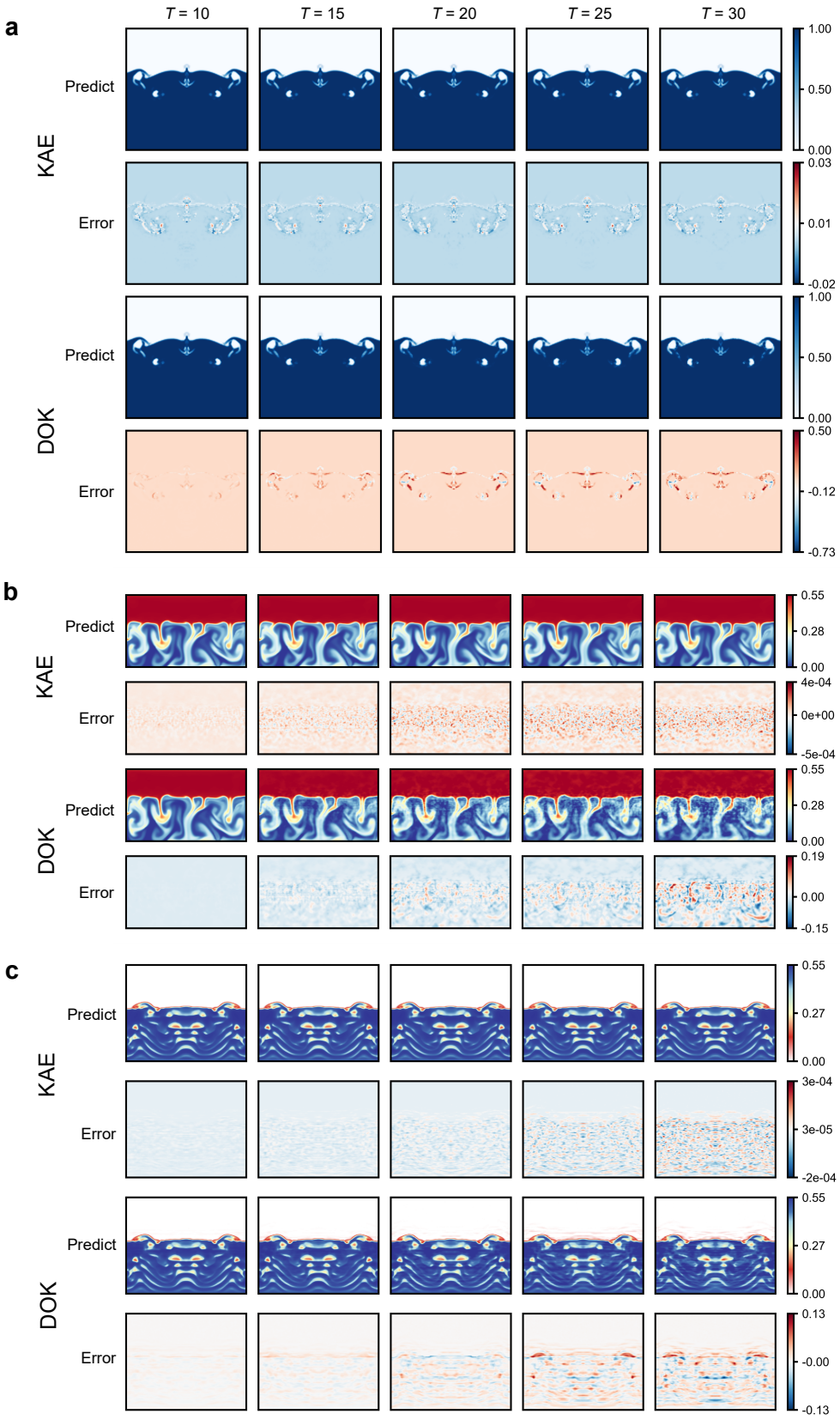

Fig. S18: Predictions of flow field evolution under different time step tests. **a** Bubble rise; **b** Particle deposition case; **c** Fluidized bed.

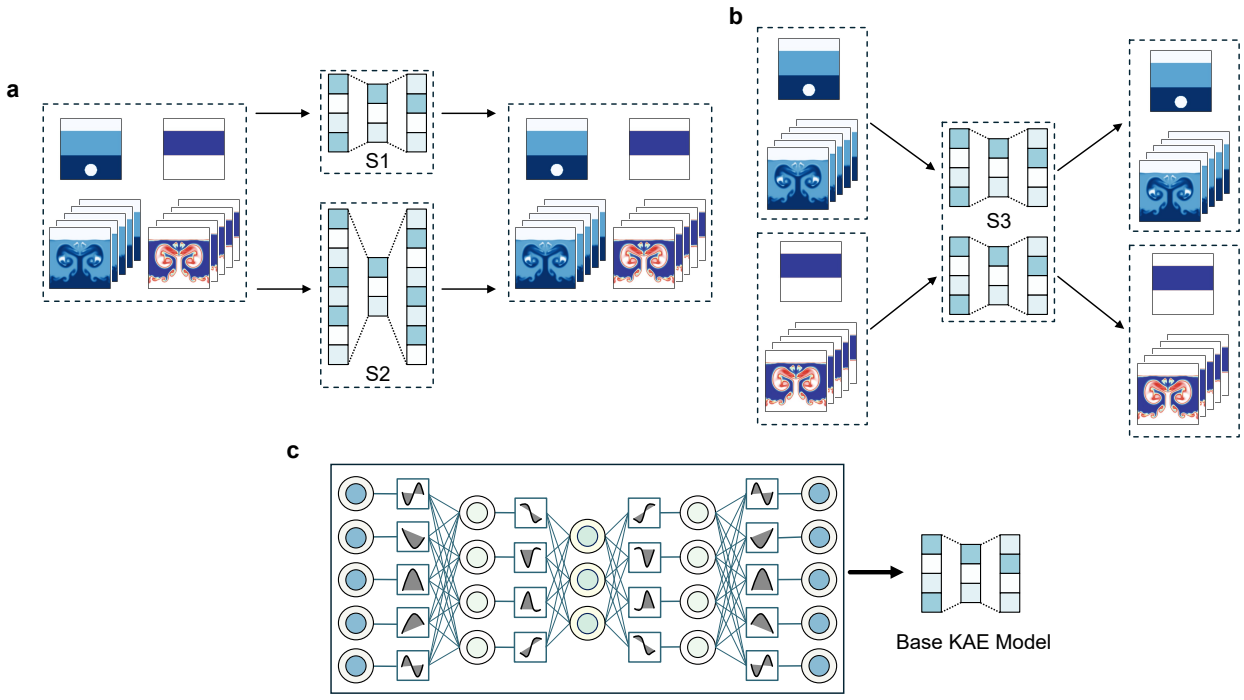

Fig. S19: KAE training strategies for multiphase flow.

accuracy decreases most markedly, particularly between 20 and 30 steps, where striped high-error bands emerge at gas-solid interfaces and internal particle textures become increasingly diffuse.

Overall, temporal-resolution densification yields limited yet stable gains for KAE. For DOK, the increase in error is not due to "accumulation" from iterative rollouts, but rather to fixed model capacity and training budget being distributed across more time points, requiring approximation over a broader temporal spectrum and more complex spatiotemporal couplings. Interface-dominated flows with simple topology, such as bubble rise, remain relatively robust at higher resolutions, whereas vortex-dominated particle deposition and strongly multiscale fluidized bed flows contain richer high-frequency information, making them harder to fit and more prone to error growth.

F Framework extension: Multiphase flow applications

In the preceding experiments, the focus was on two-phase flows, where the prediction target was the temporal evolution of a single phase's volume fraction. To address multiphase ($N_\phi > 2$), flow problems, the baseline model architecture must be extended to handle multi-input/multi-output tasks. Taking a three-phase flow as an example, the goal is to simultaneously predict the temporal evolution of two-phase volume fraction fields.

For the KAE, we designed three dimensionality-reduction and reconstruction strategies (S1-S3) for comparison (Fig. S19):

Strategy S1: A single baseline KAE (Table S1) is used to encode and decode both phase volume-fraction fields, i.e., the same encoder/decoder weights process all phase data. This maps different

Table S4: Model parameters for different KAE strategies.

Strategy	Parameters (M)	FLOPs (M)	Epoch time (s)
S1	166 M	333 M	4.4 s
S2	333 M	666 M	6.6 s
S3	333 M	333 M	5.6 s

phases into a shared latent space to extract cross-phase structural features.

Strategy S2: Builds upon S1 by increasing the baseline KAE's hidden-layer width from 1008 to 2016 (latent dimension unchanged) to enhance representation capacity for multiphase complexity. Phases still share a single latent space, but the wider network can capture more intricate relationships.

Strategy S3: Constructs an independent KAE (identical to the baseline architecture) for each phase, each learning its own latent representation. This gives each phase an independent latent space, avoiding cross-phase interference but preventing direct capture of shared features. Two KAEs must be trained, but without parameter count or computational overhead exceeding S1.

For the DOK, all strategies adopt the same multi-input/multi-output design shown in Fig. S2. Two independent branch nets extract the initial latent vectors for the respective phases, which are then fused via element-wise multiplication into a combined initial feature. This is merged with trunk-net temporal features to produce the latent vector for the target time. Finally, the corresponding KAE decoder reconstructs the two-phase volume-fraction fields at each predicted time.

The training results for each strategy are shown in Fig. S20, with prediction errors reported in Fig. 8 of the main text. With the DOK structure fixed, all three KAE strategies achieve high-fidelity reconstruction and temporal prediction for both phases. Prediction errors are similar overall, indicating comparable global accuracy; however, in high-gradient regions such as interface details and local vorticity, the errors follow the trend S1 $>$ S2 $>$ S3. S1 is limited by shared latent-space capacity, making it harder to capture fine-scale features of both phases simultaneously. S2 alleviates this limitation through width expansion, though some interference remains. S3, by assigning each phase its own latent space, minimizes local error and maximizes physical consistency while matching S1's parameter count and computational cost, achieving the best overall accuracy.

As summarized in Table S4, S1 offers the shortest training time at 4.4 s per epoch due to minimal parameter count and FLOPs, making it suitable for rapid prototyping or resource-constrained scenarios. S2 balances accuracy and capacity but increases training time to 6.6 s per epoch, reflecting the combined linear and quadratic costs of width scaling. S3 matches S2's parameter count but, owing to independent branches and a fixed 333 M FLOPs, trains in 5.6 s per epochabout 15% faster than S2—making it advantageous for large-scale concurrent inference and online updates.

In summary, if maximum local accuracy and physical consistency are the priority and moderate training cost is acceptable, S3 is recommended. If constrained by memory or real-time requirements,

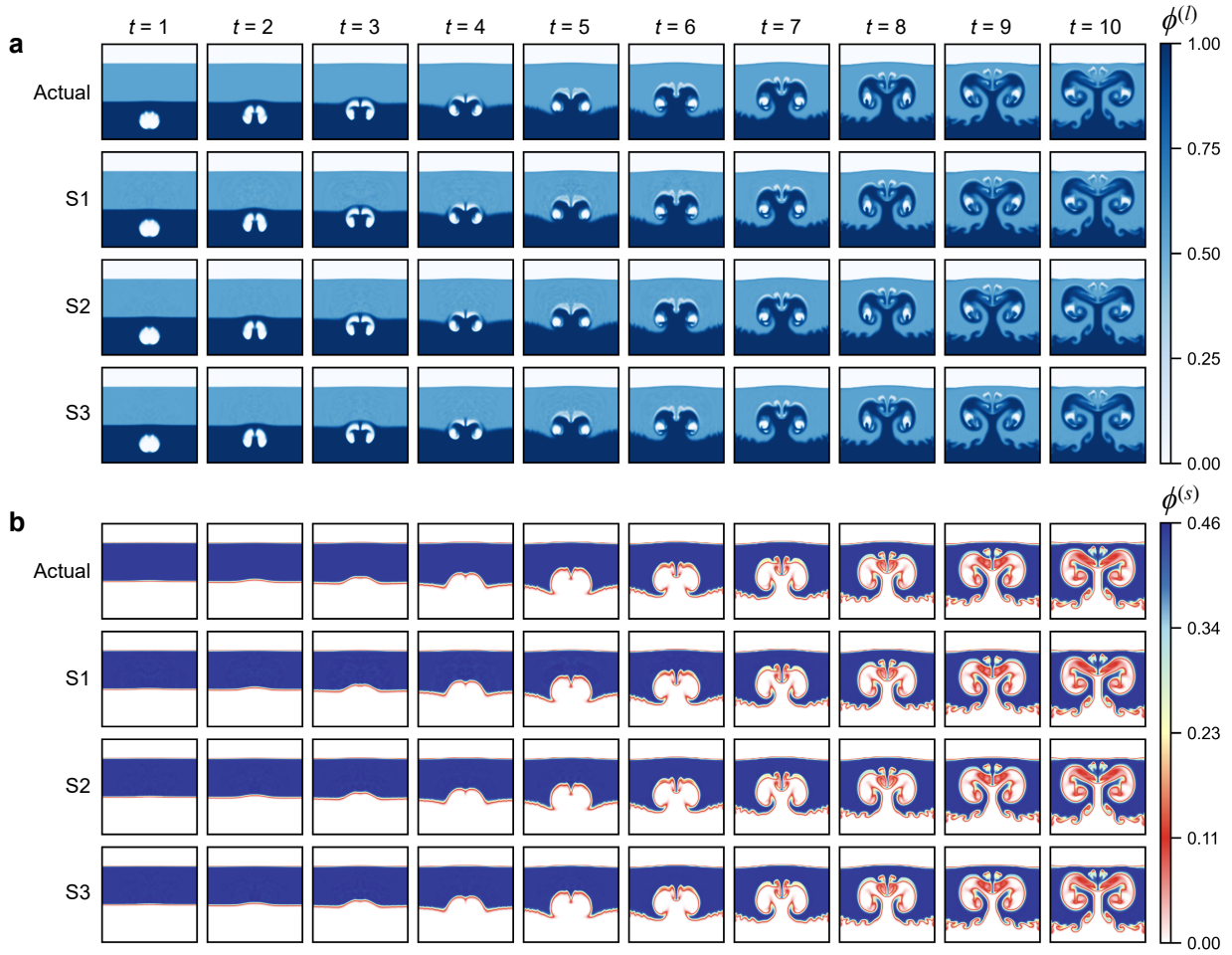

Fig. S20: Predictions of flow field evolution multiphase cases under different KAE training strategies. The snapshots at time frames for **a** Bubble rise; **b** Particle deposition case; **c** Fluidized bed.

S1 enables rapid iteration. S2 is appropriate when both accuracy and shared latent-space alignment are important and computational resources are sufficient.

References

[1] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, June 2024.

[2] Ziming Liu, Pingchuan Ma, Yixuan Wang, Wojciech Matusik, and Max Tegmark. Kan 2.0: Kolmogorov-arnold networks meet science, August 2024.

[3] Fangchen Yu, Ruilizhen Hu, Yidong Lin, Yuqi Ma, Zhenghao Huang, and Wenye Li. Kae: Kolmogorov-arnold auto-encoder for representation learning, December 2024.

[4] Diab W. Abueidda, Panos Pantidis, and Mostafa E. Mobasher. Deepokan: Deep operator

network based on kolmogorov arnold networks for mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 436:117699, March 2025.

[5] Amanda A. Howard, Bruno Jacob, Sarah H. Murphy, Alexander Heinlein, and Panos Stinis. Finite basis kolmogorov-arnold networks: Domain decomposition for data-driven and physics-informed problems, June 2024.

[6] Longlong Li, Yipeng Zhang, Guanghui Wang, and Kelin Xia. Kolmogorov–arnold graph neural networks for molecular property prediction. *Nature Machine Intelligence*, August 2025.

[7] R.W Barnard, G Dahlquist, K Pearce, L Reichel, and K.C Richards. Gram polynomials and the kummer function. *Journal of Approximation Theory*, 94(1):128–143, July 1998.

[8] Germund Dahlquist and Åke Björck. *Numerical Methods in Scientific Computing*. Society for Industrial and Applied Mathematics, Philadelphia, 2008.

[9] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. In Lior Rokach, Oded Maimon, and Erez Shmueli, editors, *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pages 353–374. Springer International Publishing, Cham, 2023.

[10] Vivek Oommen, Khemraj Shukla, Somdatta Goswami, Rémi Dingreville, and George Em Karniadakis. Learning two-phase microstructure evolution using neural operators and autoencoder architectures. *npj Computational Materials*, 8(1):190, September 2022.

[11] Alberto Solera-Rico, Carlos Sanmiguel Vila, Miguel Gómez-López, Yuning Wang, Abdulrahman Almashjary, Scott T. M. Dawson, and Ricardo Vinuesa. β-variational autoencoders and transformers for reduced-order modelling of fluid flows. *Nature Communications*, 15(1):1361, February 2024.

[12] Tongfei Liu, Jianjian Xu, Tao Lei, Yingbo Wang, Xiaogang Du, Weichuan Zhang, Zhiyong Lv, and Maoguo Gong. Aekan: Exploring superpixel-based autoencoder kolmogorov-arnold network for unsupervised multimodal change detection. *IEEE Transactions on Geoscience and Remote Sensing*, 63:1–14, 2025.

[13] Mohammadamin Moradi, Shirin Panahi, Erik Bollt, and Ying-Cheng Lai. Kolmogorov-arnold network autoencoders, October 2024.

[14] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021.

[15] Junyan He, Seid Koric, Shashank Kushwaha, Jaewan Park, Diab Abueidda, and Iwona Jasiuk. Novel deeponet architecture to predict stresses in elastoplastic structures with variable complex

geometries and loads. *Computer Methods in Applied Mechanics and Engineering*, 415:116277, October 2023.

[16] Sharmila Karumuri, Lori Graham-Brady, and Somdatta Goswami. Physics-informed latent neural operator for real-time predictions of complex physical systems, June 2025.

[17] Katiana Kontolati, Somdatta Goswami, George Em Karniadakis, and Michael D. Shields. Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems. *Nature Communications*, 15(1):5101, June 2024.

[18] Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

[19] Jiahao Zhang, Shiqi Zhang, and Guang Lin. Multiauto-deeponet: A multi-resolution autoencoder deeponet for nonlinear dimension reduction, uncertainty quantification and operator learning of forward and inverse stochastic problems, April 2022.

[20] Alexander Dylan Bodner, Antonio Santiago Tepsich, Jack Natan Spolski, and Santiago Pourteau. Convolutional kolmogorov-arnold networks, November 2024.

[21] Ivan Drokin. Kolmogorov-arnold convolutions: Design principles and empirical studies, July 2024.

[22] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Alykhan Tejani, Johannes Totz, Christian Ledig, and Zehan Wang. Is the deconvolution layer the same as a convolutional layer?, 2016.