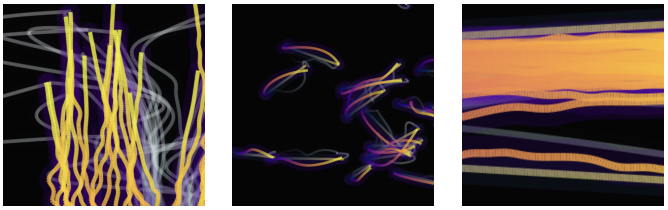
PIVONet: A Physically-Informed Variational Neuro ODE Model for Efficient Advection-Diffusion Fluid Simulation

Hayson Cheung*, Ethan Long†, David Lin‡

*1010907823, †1010941509, ‡1010900759

***Abstract*—We present PIVONet (*P*hysically-*I*nformed *V*ariational *O*DE *N*eural *Net*work), a unified framework that integrate Neural Ordinary Differential Equations (Neuro-ODEs) with Continuous Normalizing Flows (CNFs) for stochastic fluid simulation and visualization. First, we demonstrate that a physically informed model, parameterized by CNF parameters θ, can be trained offline to yield an efficient surrogate simulator for a specific fluid system, eliminating the need to explicitly simulate the full dynamics. Second, by introducing a variational model with parameters ϕ that captures latent stochasticity in observed fluid trajectories, we model the network output as a variational distribution and optimize a pathwise Evidence Lower Bound (ELBO), enabling stochastic ODE integration that captures turbulence and random fluctuations in fluid motion (advection-diffusion behaviors).**



(a) (b) (c)

Fig. 1. Generated flow trajectories overlaid onto ground truth: (a) target-matching under vortex shearing, (b) sparse particle regions, and (c) flow separation.

I. Introduction

The motion of particles in fluids is governed by complex interactions between the underlying flow field and stochastic perturbations arising from thermal effects or turbulence [1]. Traditional computational fluid dynamics (CFD) methods provide high-fidelity simulations but are often prohibitively expensive for real-time prediction or large-scale parameter studies.

Whereas our method of data collection consists of an imperfect computational simulation of particle trajectories governed by a set of flow and fluid mechanical equations, we showed that our model successfully captured flow features in non-laminar flow: advection-diffusion, turbulence, and flow separation. In turn, it is able to predict particle trajectories in two-dimensional fluid flows efficiently, reflecting its viability in modeling real-world systems such as combustion engines and atmospheric circulations.

While our model is trained on a limited set of flow conditions, it demonstrates that latent representations can capture the key patterns present within this regime. Although we do not claim broad generalization, this provides a foundation on which broader, covariate-conditioned generalization could be built in future work.

II. Background

A. Governing Advection-Diffusion Equation

We model particle trajectories using the overdamped Langevin equation, which captures advection-diffusion physics while remaining computationally tractable [2]. The position $\mathbf{x}_t = (x_t, y_t) \in \mathbb{R}^2$ evolves according to:

$$d\mathbf{x}_t = \mathbf{u}(\mathbf{x}_t)\, dt + \sqrt{2D}\, d\mathbf{W}_t \tag{1}$$

This is a SDE where $\mathbf{u}(\mathbf{x}_t)$[1] is the local fluid velocity field (Fig. 2a), D is the diffusion coefficient, and $\mathbf{W}_t$ is a two-dimensional Brownian motion, neglecting inertial effects.

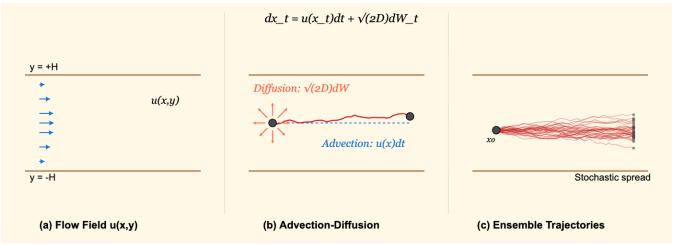


Fig. 2. **Langevin Dynamics: Particle Motion in Fluid Flow.** The Langevin equation $dx_t = u(x_t)dt + \sqrt{2D}dW_t$ governs particle motion in fluid flows. (a) Poiseuille velocity profile $u(x, y)$ with parabolic shape. (b) Single particle trajectory showing deterministic advection (blue dashed) with superimposed stochastic Brownian fluctuations (red solid). (c) Multiple stochastic realizations from the same initial condition x_0, illustrating the spread of particle positions due to randomness.

The SDE in Eq. (1) decomposes particle motion into deterministic advection along streamlines and stochastic diffusion (Fig. 2b). The equation is discretized by Euler-Maruyama:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{u}(\mathbf{x}_t)\, \Delta t + \sqrt{2D\, \Delta t}\, \boldsymbol{\xi}, \quad \boldsymbol{\xi} \sim \mathcal{N}(0, \mathbf{I}) \tag{2}$$

with reflecting boundaries at $y = \pm H$. It is mathematically analogous to lattice diffusion systems expressed as linear ODEs of the form

$$\mathbf{x}'(t) = kA\mathbf{x}(t) \tag{3}$$

[1]The simulation to obtain u will be discucsed in Section V-B

with reflecting endpoints, whose eigenvalue structure governs long-term equilibrium behavior [3].

The stochastic nature of this process leads to variability in particle trajectories even from identical initial conditions (Fig. 2c), consistent with classical dispersion theory [4]. For each trajectory, we record initial conditions $\mathbf{x}_0$, flow parameters α, and the sequence $\{\mathbf{x}_t\}_{t=0}^{T}$.

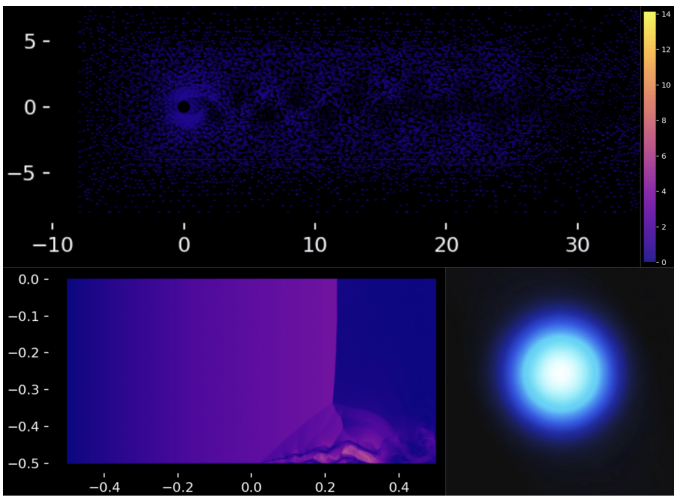


Fig. 3. **Flow Profiles for Trajectory Generation.** (a) and (b) are velocity profiles on the top and bottom right respectively. (c) is located bottom right, a pressure profile visualized via Paraview [5]. (a) *Shock Tube* configuration showing discontinuity complex behavior near the wall. (b) *Cylinder Flow*, where the flow impinges on a cylinder, producing a radial velocity distribution and oscillatory trail downstream. (c) *Euler Vortex*, illustrating rotational shear flow with centerline velocity $U_{\max}$.

B. Normalizing Flow

For trajectory modeling, continuous normalizing flows (CNFs) treat the transformation as an ODE [6], [7]. This builds upon earlier discrete flow-based generative models such as NICE [8], RealNVP [9], and Glow [10], originally introduced for density estimation and variational inference [11].

$$\frac{d\mathbf{z}(t)}{dt} = g_{\boldsymbol{\theta}}(\mathbf{z}(t), t), \quad \mathbf{z}(0) = \mathbf{z}_0, \quad \mathbf{z}(1) = \mathbf{x} \tag{4}$$

The log-density evolves according to the instantaneous change-of-variables formula [6]:

$$\frac{d \log p(\mathbf{z}(t))}{dt} = -\mathrm{tr}\left(\frac{\partial g_{\boldsymbol{\theta}}}{\partial \mathbf{z}}\right) \tag{5}$$

This equation tracks how probability mass compresses or expands as particles flow through the learned transformation. The trace of the Jacobian measures the divergence of the velocity field g_{ϕ}: positive divergence (expansion) decreases density, while negative divergence (compression) increases it. For our particle trajectories, this enables the model to learn where paths concentrate (high-probability regions like flow streamlines) versus where they disperse (low-probability regions affected by strong diffusion) [7].

C. Bi-directionality and Training

We model the mappings between the two distributions as a diffeomorphism. Hence, a key advantage of CNF is bi-directionality: we can integrate the ODE system both forward (from simple $\mathbf{z}_0 \sim \mathcal{N}(0, \mathbf{I})$ to complex trajectory distribution) and backward (from observed trajectory to its latent representation). The forward process generates new trajectories by sampling, while the backward process computes exact likelihoods for training [11]. Figure 4 illustrates this bidirectional transformation, showing how the initially Gaussian distribution evolves through intermediate time steps into the complex, multimodal distribution of particle trajectories conditioned on flow parameters. The ODE system in Eqs. (4)–(5) is solved jointly using adaptive solvers, allowing us to compute exact likelihoods for any trajectory under the learned distribution—essential for training and uncertainty quantification [10].

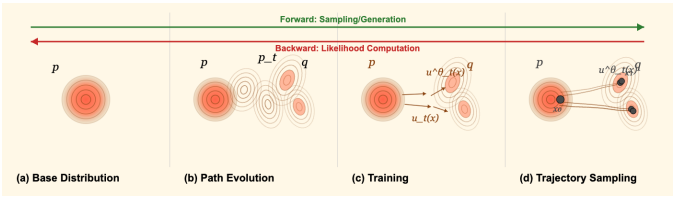


Fig. 4. **Continuous Normalizing Flow for Particle Trajectory Prediction .** The CNF transforms a simple Gaussian base distribution p (a) through intermediate distributions (b) via learned velocity fields $u_t(x)$ (c) to match the complex trajectory distribution q. During sampling (d), particles follow stochastic paths from initial position x_0 to predicted position x_1, capturing both deterministic flow and diffusive behavior.

We condition on $\mathbf{x}_0$ and flow parameters $\{U_0, \gamma, U\max, H, D, \ldots\} \subset \boldsymbol{\alpha}$ via concatenation or feature-wise transformations. The model is trained by maximizing log-likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{x}_0, \boldsymbol{\alpha})\right] \tag{6}$$

For inference, we sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ and apply $\hat{\mathbf{x}} = f_{\boldsymbol{\theta}}(\mathbf{z}; \mathbf{x}_0, \boldsymbol{\alpha})$ to obtain predicted trajectories without re-running simulations. This framework captures both deterministic flow-driven behavior and stochastic variability, enabling probabilistic trajectory forecasting.

III. Model

A. Neural Differential Equation Backbone

We train a continuous normalizing flow (CNF) to inherit the backbone dynamics of the flow [6], [7]. The CNF delivers an expressive drift $f_\theta(z)$ that encodes the physics of the training dataset, while the VSDE extends it with stochastic diffusion and posterior controls to match observed trajectories. We treat the CNF as a frozen module and learn only the additional control signal $u(z, t)$ and diffusion coefficient g, which keeps the base physics intact while providing flexibility during inference. Adding the VSDE path is therefore equivalent to augmenting the modeling ODE with a nonhomogeneous stochastic forcing

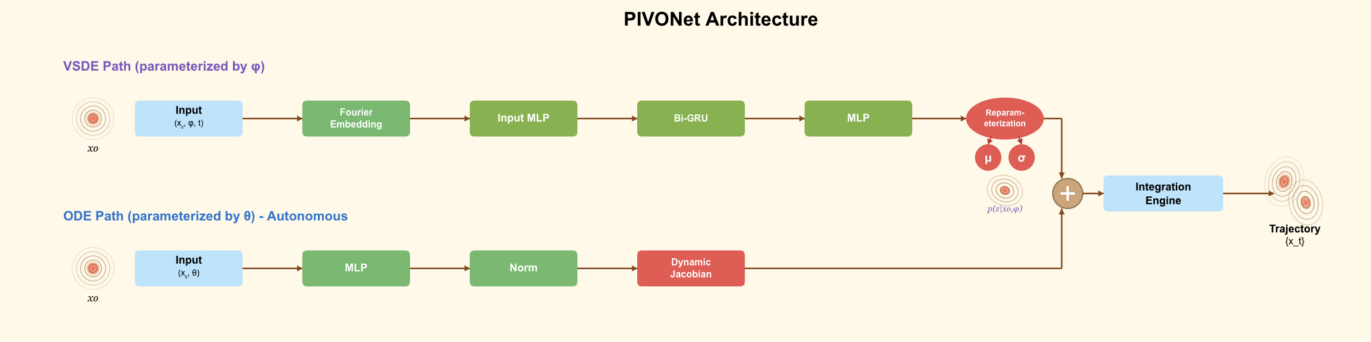


Fig. 5. The VSDE framework couples a frozen CNF backbone with learned posterior controls and physics-informed losses.

term, letting the learned controls and diffusion absorb dataset-specific deviations without corrupting the pretrained drift.

$$\mathrm{d}z = f_\theta \,\mathrm{d}t + g_\phi \,\mathrm{d}W_t \tag{7}$$

Here the control $u(z,t)$ and diffusion coefficient g act as the learned forcing, while $\mathrm{d}W_t$ denotes the Brownian increment that encodes stochasticity.

By adding the varieational component, our model tend to explore different pathways when reconstructing the same input; This results in a higher stablity as the model is more likely to stay on the learned manifold, as the varieatioanl component can help nudge the trajectory back when it starts to drift away.

B. Architecture

Input trajectories are first encoded by the Trajectory Encoder, which produces a compact posterior context vector. This context initializes a surrogate posterior $q(z_0 \mid x)$ and conditions the Posterior Drift Net. The posterior drift network, implemented as an MLP with Fourier time embeddings, outputs both the control vector u and optional diffusion corrections [12]. The VSDE integrator then combines the fixed CNF drift, the learned control, and the Brownian diffusion to produce latent trajectories that are mapped back to the original space for reconstruction and diagnostics (see Figure 5).

C. Neural Networks

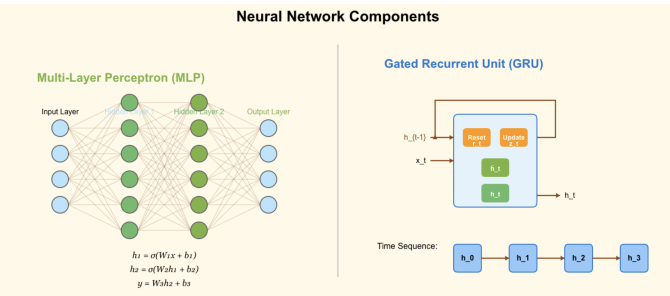


Fig. 6. Diagram of the Multi-Layer Perceptron (MLP) used for controls and the Gated Recurrent Unit (GRU) encoder that summarizes trajectories.

The neural networks in VSDE are responsible for translating data into controls and summarizing temporal context. Figure 6 highlights both modules.

1) Multi-Layer Perceptron: The control network is a shallow but wide feed-forward network known as a Multi-Layer Perceptron (MLP). Each layer multiplies its inputs by learned weights, adds biases, and applies nonlinearities so the network can represent complex control policies. The MLP consumes the concatenated tuple $(z, ctx, \text{time emb})$ and outputs the controls. Residual connections inside the MLP help stabilize training when the control signal remains small. We can express the MLP as

$$u = \sigma_L(W_L\sigma_{L-1}(\cdots\sigma_1(W_1x + b_1) + b_{L-1}) + b_L), \tag{8}$$

where $x = (z, ctx, \text{tizme emb})$, σ_i are nonlinearities activation functions, and $\{W_i, b_i\}$ are learned weights and biases for layer i. Training minimizes the joint loss $\mathcal{L}$ described above, so each parameter is updated via a gradient step such as

$$W_i^{(l)} \leftarrow W_i^{(l)} - \eta\nabla_{W_i^{(l)}}\mathcal{L}, \qquad b_i^{(l)} \leftarrow b_i^{(l)} - \eta\nabla_{b_i^{(l)}}\mathcal{L}, \tag{9}$$

where η is the learning rate from the optimizer. The gradients are computed via backpropagation through the layered MLP using the chain rule, which propagates loss signals backward through each nonlinear activation [12]; in practice we rely on the Adam optimizer [13] so the effective update also tracks first- and second-moment estimates of these gradients to stabilize and accelerate convergence.

2) Gated Recurrent Unit: The trajectory encoder uses a Gated Recurrent Unit (GRU), which is a type of recurrent network tailored for sequences [14]. It processes the observation sequence one timestep at a time, updating its hidden state and deciding how much past information to keep via gating mechanisms. This allows the GRU to remember long-range dependencies without exploding gradients and to focus on recent changes when necessary, which is essential when the VSDE must condition on hours of simulated flow. The GRU

update is guided by reset r and update z gates

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \tag{10a}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \tag{10b}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1})), \tag{10c}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{10d}$$

IV. Activation Function

The VSDE control MLP uses Gaussian Error Linear Units (GELU) in each hidden layer, which blend linear and nonlinear behaviors and keep gradients smooth so the control signal changes gradually with the latent state. The GRU encoder relies on sigmoid activations for the reset and update gates and a tanh nonlinearity for the candidate hidden state; this combination ensures gate outputs stay in $[0, 1]$ and that the recurrent dynamics remain bounded while still allowing nonlinear transformations of the inputs. A more detailed explanation of GRUs and MLP can be found in Appendix D, along with why these activations are preferred for time-series modeling and function approximation.

A. Variational Inference

We perform variational inference by sampling $n_{\text{particles}}$ latent initializations from $q(z_0 \mid x)$ and integrating each sample through the VSDE [11]. The ELBO decomposes into a reconstruction likelihood (driven by observation noise σ_{obs}), a KL divergence that anchors the posterior to a standard normal, and a control cost that penalizes large control efforts. The energy-based penalties described below are added to this objective to shepherd the latent trajectories toward physically consistent behavior. Sampling uses the reparameterization trick:

$$z_0 = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \tag{11}$$

which lets gradients flow through the encoder parameters (μ, σ) when we backpropagate the loss. The gradient of the ELBO with respect to the parameters θ is

$$\nabla_\theta \mathcal{L} = \mathbb{E}_{q(z_0|x)} \left[\nabla_\theta \log p(x \mid z) - \lambda_{\text{KL}} \nabla_\theta \log q(z_0 \mid x) + \cdots \right], \tag{12}$$

where the ellipsis hides control and physics loss derivatives and the expectation is approximated with our particles. The adjoint method ensures the gradients through the VSDE integrator remain tractable.

B. Training

1) Optimizing the Path: We backpropagate through the integrator using the adjoint sensitivity method built into the CNF. The VSDE integrator is differentiable because we keep the time grid fixed and apply deterministic updates before adding the diffusion term. The optimizer updates the Posterior Drift Net weights, the diffusion log-parameter, and the posterior encoder simultaneously.

2) Dynamic Jacobian: Because the CNF defines an instantaneous velocity field, we can compute the Jacobian of the combined drift with respect to z during training. This Jacobian is needed for KL and control cost terms as well as for ensuring stability when coupling the CNF with the posterior drift.

3) Physical Loss Function: We augment the ELBO with two complementary physics-aware losses so the VSDE latent paths respect conserved quantities and known PDE structure. The first loss tracks kinetic energy fluctuations by computing $\frac{1}{2}\rho\|z_{t+1} - z_t\|^2$ along each trajectory and penalizing sudden jumps that contradict the smooth advection-dominated physics; this couples adjacent timesteps through finite differences of latent velocities. The second loss enforces the residual of the compressible energy equation $\rho c_p(\partial_t T + u \cdot \nabla T) = k\nabla^2 T + \Phi$ by discretizing the temporal derivatives via backward differences and estimating spatial gradients from the CNF decoder; penalizing this residual steers the diffusion and control terms to satisfy thermodynamic balance in expectation. Each term is averaged over particles before being added to the loss, and we scale them with λ_{phys} and λ_{pde} so the model can trade reconstruction fidelity for these physical regularizers.

4) Joint Evidence Lower-Bound: The final loss is

$$\begin{aligned}\mathcal{L} = &-\mathbb{E}_{q(z_0|x)} \left[\log p(x \mid z)\right] + \lambda_{\text{KL}} \text{KL}(q(z_0 \mid x) \| p(z_0)) \\ &+ \lambda_u \mathcal{L}_{\text{control}} + \lambda_{\text{phys}} \mathcal{L}_{\text{energy}} + \lambda_{\text{pde}} \mathcal{L}_{\text{balance}}.\end{aligned} \tag{13}$$

The λ weights for the KL, control, and physics penalties are instead selected manually in this study to balance numerical stability with expressive control rather than through an exhaustive grid search.

5) Hyperparameters and Optimizers: Training uses the Adam optimizer as described in [13], with weight decay (AdamW) and cosine learning-rate annealing. The learning rate schedule is warmed up for the first 2k steps and decays toward 1e-5. Control cost scale and diffusion coefficients are initialized to encourage low-variance trajectories, and the physics weights start near zero to prevent destabilizing gradients.

C. Inference

1) Joint ODE Integration: Inference samples from the encoder to obtain posterior contexts and then integrates the VSDE for both control and diffusion to produce latent trajectories. The output is decoded via the CNF to reconstruct the observable variables while preserving uncertainty quantification through the ensemble of particles.

2) Integration Methods: We expose Euler, Heun, RK4, and Dormand–Prince integrators for the deterministic part of the dynamics. Diffusion is always attached via Euler–Maruyama (2). The inference API allows selecting the integrator and diffusion scale so experiments can trade accuracy for speed, though emprical results shows little differences in accuracy across integrators for our application.

D. Guard Rails During Inference

Special-case logic clamps controls and enforces absorbing boundaries when trajectories approach invalid states, which prevents numerical explosions. If a trajectory wanders outside the training manifold, its weight in the ELBO computation is attenuated to focus the optimization on valid samples. We formalize this by defining a soft validity indicator

$$\gamma(z) = \exp\left(-\alpha \max\left(\|z\| - R, 0\right)^2\right), \tag{14}$$

where R is a radius extracted from the training states and α controls how sharply samples are down-weighted. The encoder weight is multiplied by $\gamma(z)$ before being averaged, and the posterior control is clamped according to

$$u_{\text{clamp}} = \text{sign}(u) \min\left(|u|, u_{\max}\right), \tag{15}$$

ensuring the VSDE cannot inject unbounded drifts even when extrapolating."*** End Patch

V. Experiment Setup

A. Hypothesis

The experiment is designed to test the hypothesis that:

1) Using neuro ODE-based generative models can effectively capture complex fluid dynamics.
2) Integrating VSDE controllers with CNF backbones enhances the model's ability to represent stochastic behaviors in fluid flows.
3) The proposed framework can generalize across different flow regimes, including inviscid, viscous, and incompressible scenarios.
4) The proposed framewrok is numerically stable and can generalize with differnt integrators.

B. Data Simulation

To generate high-fidelity, time-resolved flow fields for training and evaluation of continuous normalizing flow (CNF) and stochastic differential equation (SDE) models, we employed the open-source `PyFR` framework as the numerical simulation engine [15]. `PyFR` is a Python-based computational fluid dynamics (CFD) platform that implements high-order accurate spatial discretization using the flux reconstruction approach and is capable of solving the compressible Navier–Stokes equations on unstructured meshes across heterogeneous hardware architectures.

The specifications for the `PyFR` orchestrations is in Appendix A

The governing conservation laws for compressible fluid flow are expressed as:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = \nabla \cdot \mathbf{F}_v(\mathbf{U}, \nabla \mathbf{U}), \tag{16}$$

where $\mathbf{U}$ denotes the vector of conserved variables, $\mathbf{F}$ and $\mathbf{F}_v$ represent inviscid and viscous fluxes respectively, and the right-hand side encapsulates viscous transport contributions.

Spatial discretization in `PyFR` employs a high-order flux reconstruction scheme that replaces spatial derivatives with discrete algebraic operators evaluated at solution and flux points defined on the computational mesh. Following discretization of the spatial operators, the partial differential equation (PDE) system is reduced to a large system of ordinary differential equations (ODEs):

$$\frac{d\mathbf{u}(t)}{dt} = \mathbf{f}(\mathbf{u}(t)), \tag{17}$$

where $\mathbf{u}(t)$ is the vector of all discrete flow variables comprising density, momentum, and energy at each node, and $\mathbf{f}$ encapsulates the semi-discrete flux and source approximations. `PyFR` advances this system in time using explicit time integration schemes suitable for high-order methods.

Because the resulting solutions still encode conserved quantities, they become the reference for our physics-informed loss. We later compute a transformed temperature field and velocity signal from the discrete states and penalize the residual of the thermal energy equation

$$\rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) = k \nabla^2 T + \Phi, \tag{18}$$

which balances temporal and advective heating against conduction and viscous dissipation. This augments the variational autoencoder with a PDE-based constraint that is consistent with the compressible simulation data.

C. Flow Cases

We evaluate our framework across three canonical fluid dynamics scenarios that exhibit diverse flow behaviors and complexities:

- **2D Euler Vortex:** This case involves simulating a two-dimensional vortex in an inviscid fluid governed by the Euler equations. The vortex dynamics test the model's ability to capture rotational flow features and conserve vorticity over time.
- **Viscous Shock Tube:** This scenario simulates the evolution of shock waves and contact discontinuities in a viscous medium. The shock tube problem challenges the model to accurately represent sharp gradients and dissipative effects inherent in viscous flows.
- **Incompressible Cylinder Flow:** This case examines the flow around a circular cylinder in an incompressible fluid. The cylinder flow problem assesses the model's capability to capture boundary layer development, vortex shedding, and wake dynamics.

The specific configurations for each flow case, including domain size, mesh resolution, initial and boundary conditions, and physical parameters (e.g., Reynolds number, Mach number), are detailed in the code repository accompanying this manuscript. In addtional below is a brief overview of the governing equations and numerical setups for each scenario.

a) 2D Euler Vortex: The 2D Euler vortex is governed by the inviscid Euler equations, which describe the conservation of mass, momentum, and energy in a compressible fluid without viscosity. The governing equations are expressed in conservation form as:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0, \tag{19}$$

where $\mathbf{U}$ is the vector of conserved variables, and $\mathbf{F}(\mathbf{U})$ represents the flux tensor. The initial condition consists of a Gaussian vortex superimposed on a uniform flow field. The computational domain is discretized using a structured mesh with periodic boundary conditions.

b) Viscous Shock Tube: The viscous shock tube problem is governed by the compressible Navier–Stokes equations, which account for viscous effects in addition to the conservation of mass, momentum, and energy. The governing equations are given by:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = \nabla \cdot \mathbf{F}_v(\mathbf{U}, \nabla \mathbf{U}), \tag{20}$$

where $\mathbf{F}_v(\mathbf{U}, \nabla \mathbf{U})$ represents the viscous flux tensor. The initial condition consists of a high-pressure region adjacent to a low-pressure region, separated by a diaphragm. The computational domain is discretized using an unstructured mesh with reflective boundary conditions at the tube walls.

c) Incompressible Cylinder Flow: The incompressible cylinder flow is governed by the incompressible Navier–Stokes equations, which describe the conservation of mass and momentum in an incompressible fluid. The governing equations are expressed as:

$$\nabla \cdot \mathbf{u} = 0, \tag{21}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}, \tag{22}$$

where $\mathbf{u}$ is the velocity field, p is the pressure field, and ν is the kinematic viscosity. The initial condition consists of a uniform flow field impinging on a circular cylinder. The computational domain is discretized using an unstructured mesh with no-slip boundary conditions on the cylinder surface and far-field conditions at the domain boundaries.

D. Experiment Workflow

The empirical pipeline described below is executed across several flow cases (2D Euler vortex, viscous shock tube, and incompressible cylinder) whose configurations are captured in the code base. Appendix B details the software implementation, which includes clearly defined wroflow modules that matches the steps below:

1) **Trajectory simulation:** 4096 particles are marched for 240 steps with varying Δt to produce bundles of trajectories, velocity snapshots, and mesh coordinates that subsequent modules reuse.
2) **CNF backbone training:** The CNF drift network trains on the trajectory endpoints in minibatches of 512 for eight epochs, using a 0.002 learning rate, hidden dimension 64, depth 3, and context dimension 3;
3) **VSDE controller training:** With the CNF frozen, the variational posterior drift net and diffusion scale optimize over 50 epochs with batch size 2048, learning rate 0.01, 64 particles, and 120 integration steps; control cost, kinetic-energy, and PDE penalties regularize the latent paths while the diffusion remains learnable.
4) **Inference diagnostics:** Ensembles of 64 particles over 120 steps are decoded through the CNF+VSDE pair to produce overlay PNGs (VSDE vs. GT, CNF vs. GT, mixed overlays) and velocity-phase plots. We save the experimental artifacts.
5) **Velocity visualization:** A 50k-vector sample renders the velocity phase-space PNG, and a quiver animation (vector stride 3, 12 fps) animates temporal evolution for QA.

E. Training and Experimental Metrics

Training runs used Apple M1 silicon (MPS backend) so the CNF and VSDE modules finish within roughly 15 minutes each on the cached trajectory bundles; we warm up the learning rate for 1k steps and rely on AdamW with cosine annealing to stabilize convergence. The CFD trajectories are split into training and validation bundles that emphasize the vortex phase so the learned dynamics focus on the advection-dominated regime.

Hyperparameter values are summarized in Table II (Appendix E): the CNF uses minibatches of 512 for eight epochs with a 0.002 base learning rate, while the VSDE controller executes 50 epochs with batch size 2048, learning rate 0.01, 64 posterior particles, 120 integration steps, and physics penalties on control cost, kinetic energy, and the PDE residual.

F. Evaluation Sets

1) VSDE vs. CNF-only Dynamics: We measure reconstruction accuracy on held-out bundles for each flow case and report the VSDE-augmented latent paths alongside the baseline CNF-only ODE. Across the Euler vortex, viscous shock tube, and incompressible cylinder experiments, the VSDE consistently improves RMSE and likelihood by steering trajectories back toward the learned manifold whenever the CNF starts to deviate.

2) Integrator Comparison: Inference stability is confirmed across Euler, Heun, RK4, and Dormand–Prince integrators for each flow, using overlay plots and energy statistics to quantify fidelity. After tuning the VSDE diffusion scale, all integrators reach similar accuracy, which confirms that the learned control dominates ensemble behavior while the choice of deterministic integrator only marginally affects fidelity in these flow regimes.

VI. Results

This section presents experimental results corresponding to the hypotheses outlined. As well, MAE is chosen as the primary metric for evaluating model performance across various scenarios. This is becasue:

1) **Interpretability**: MAE provides a straightforward measure of average error in the same units as the predicted variable, making it easy to interpret the model's performance in practical terms.
2) **Linearity**: MAE treats all errors equally, providing a linear measure of average error without disproportionately penalizing larger errors, which is suitable for many fluid dynamics applications where consistent performance is desired.
3) **Robustness to Outliers**: While MAE is sensitive to outliers, it is less so than metrics like Mean Squared Error (MSE). This balance makes MAE a good choice for fluid dynamics data, which may contain occasional anomalies due to turbulence or measurement noise.

A. Expressiveness of ODE Backbones for Fluid Dynamics

We first evaluate the performance of different ODE-based backbone models in capturing fluidic dynamics. Overall, the ODE-only approach is able to learn the dominant, mean flow behavior and produces smooth, physically consistent trajectories that align with the large-scale structure of the flow. In particular, the homogeneous CNF successfully converges to a stable solution that reflects the primary deterministic dynamics of the incompressible cylinder flow.

However, this same determinism becomes a limitation when modeling stochastic fluid behavior. Although the model implicitly learns a probability distribution over trajectories, the absence of latent variables restricts its expressive capacity. As a result, the learned dynamics collapse toward a single dominant mode, making the model overly conservative and preventing it from capturing the full range of stochastic variations present in the system. This leads to an underrepresentation of flow variability and fine-scale randomness observed in the ground truth data. The inference results of the ODE-only model are shown in Figure 7.

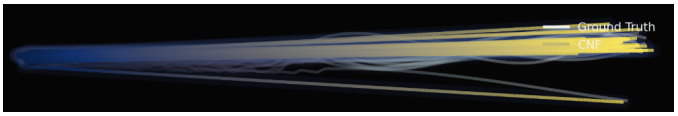


Fig. 7. Inference results of the ODE-only CNF model on the incompressible cylinder flow. Predicted trajectories are overlaid on the ground truth flow realizations. While the model accurately captures the overall mean flow structure and produces smooth, stable trajectories, the predictions collapse toward a deterministic solution. This behavior highlights the model's inability to represent stochastic variations and multi-modal flow behavior, resulting in limited diversity compared to the ground truth dynamics.

B. Comparison of CNF with and without VSDE forcing

By incorporating a learned stochastic inhomogeneous forcing term through the the entire PIVONet framework, we observe a significant enhancement in the model's ability to capture the inherent variability and randomness of fluid dynamics. The VSDE component introduces latent variables that enable the model to represent a richer distribution over possible flow trajectories, allowing it to account for fine-scale fluctuations and multi-modal behaviors that were previously unrepresented in the ODE-only approach. This results in a more faithful reproduction of the stochastic characteristics observed in the ground truth data, as the

VII. Conclusion

A. Further Disccussion

B. Summary of Contributions

The project aims to model physical representations with efficient generative models, making it suitable and effective for applications in fluid modeling.

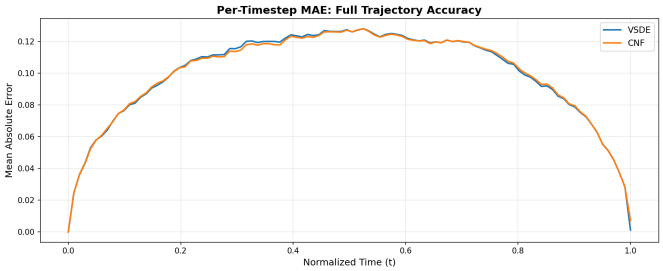


Fig. 8. Caption

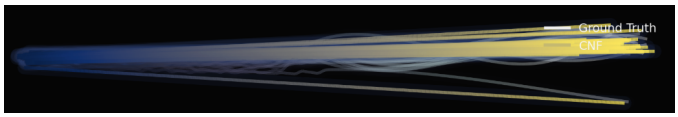


Fig. 9. Caption

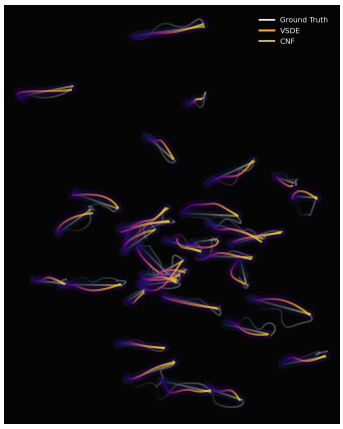


Fig. 10. Caption

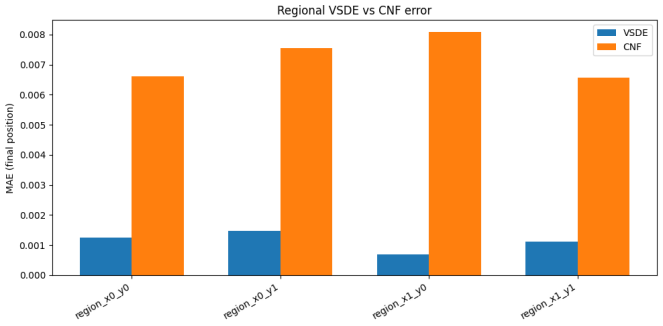


Fig. 11. Like different regimes

C. Future Works

References

[1] T. M. Squires and S. R. Quake, "Microfluidics: Fluid physics at the nanoliter scale," *Reviews of Modern Physics*, vol. 77, pp. 977–1026, 2005.

[2] D. L. Ermak and J. A. McCammon, "Brownian dynamics with hydrodynamic interactions," *Journal of Chemical Physics*, vol. 69, pp. 1352–1360, 1978.

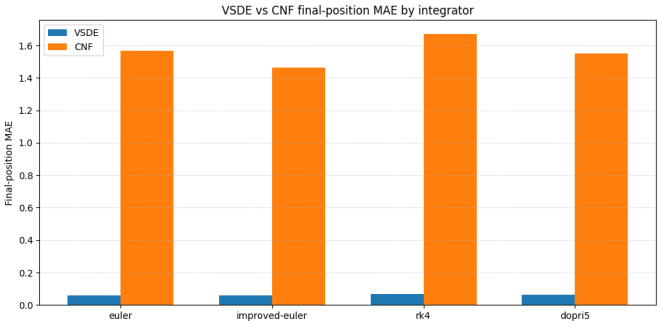


Fig. 12. Like integrator

[3] R. L. Devaney and S. G. Krantz, *Differential Equations: An Introduction to Modern Methods*. Brooks/Cole, 2nd ed., 2007.

[4] G. I. Taylor, "Dispersion of soluble matter in solvent flowing slowly through a tube," *Proceedings of the Royal Society of London. Series A*, vol. 219, pp. 186–203, 1953.

[5] Ahrens, James and Geveci, Berk and Law, Charles, "Paraview." https://www.paraview.org/, 2005. Version 5.11.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[7] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," in *Proc. International Conference on Machine Learning (ICML)*, 2019.

[8] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," in *ICLR Workshop Track*, 2014.

[9] L. Dinh, J. Sohl-Dickstein, and Y. Bengio, "Density estimation using real nvp," in *Proc. International Conference on Learning Representations (ICLR)*, 2017.

[10] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[11] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. 32nd International Conference on Machine Learning (ICML)*, 2015.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2015.

[14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014.

[15] F. D. Witherden, B. C. Vermeire, P. E. Vincent, *et al.*, "PyFR v2.0.3: Towards industrial adoption of scale-resolving simulations," *Computer Physics Communications*, vol. 311, p. 109567, 2025.

APPENDIX A
PYFR ORCHESTRATION

Simulations were orchestrated within a Jupyter Notebook environment to facilitate reproducible execution, automated workflow control, GPU acceleration via CUDA, and seamless integration with downstream post-processing and dataset management procedures.

The output of the simulation pipeline is a set of ground-truth physical trajectories representing the evolution of flow variables over time. Each simulation is initialized from a mesh file (`.msh`) defining the domain discretization and a solver configuration file (`.ini`) specifying physical parameters and temporal integration settings; these inputs are converted into `PyFR`'s internal formats prior to execution. Post-processing converts the solver state outputs into structured numerical arrays that preserve spatial and temporal coherency, yielding discrete samples of an underlying continuous-time dynamical system governed by Eq. (16). These datasets provide the reference trajectories used to train and validate the learned dynamical models [15].

To support reproducibility, all data, simulation scripts, and configuration files are provided alongside the accompanying GitHub repository.

APPENDIX B
IMPLEMENTATION AND REPRODUCIBILITY

All code lives in `github.com/haysonc/pivonet`. Clone the repository, install dependencies, and run the VSDE model with the following steps:

1) Set up the virtual environment and install the requirements:

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

2) Use the `pivo` CLI to orchestrate training/inference or call the flow module:

```
pivo
```

Dataset from `PyFR` is uploaded on Google Drive: `https://drive.google.com/file/d/1MWayzNu9oW745n9ZPRmS7aWnw8wthX8k/view?usp=drive_link`. You can download and extract it into the `data/` folder in the repository root.

A sample repository stucture is as follows:

```
pivonet/
|-- data/
|   |-- 2d-euler-vortex/
|   `-- ...
|-- src/
|   |-- models/
|   |-- utils/
|   `-- ...
`-- ...
```

For more detailed instructions, refer to the `README.md` in the repository root. The code style is enforced with `ruff` and is configured via `pyproject.toml` ; all configuration values follow YAML-like semantics with leading keys for clarity, so keep any new scripts aligned with those defaults.

APPENDIX C
GLOSSARY OF TECHNICAL TERMS

See Table I next page for selected terms and phrases that appear in the main paper and are commonplace in machine learning and physics-informed neural networks (PINNs) literature. This glossary is intended to help readers unfamiliar with these concepts better understand the terminology used throughout this work.

TABLE I
GLOSSARY OF KEY VSDE, PINN, AND ML CONCEPTS REFERENCED IN THIS WORK.

Latent Representations	
Variational parameters	The encoder output pair μ, σ that parametrize the surrogate posterior $q(z_0 \mid x)$; sampled via reparameterization trick to allow gradient flow.
Posterior	Distribution over latent initial states conditioned on observations; VSDE trains $q(z_0 \mid x)$ to maintain data consistency.
Latent space	Lower-dimensional space where the CNF and VSDE operate; latent coordinates z are decoded back to observed data x.
Latent vector	Hidden representation z in latent space; VSDE integrates z to produce trajectories decoded into observations.
Manifold	Lower-dimensional surface in latent space where training samples lie; CNF and VSDE enforce drift along it.
Reparameterization trick	Method to sample from distributions in a differentiable way by expressing samples as deterministic functions of parameters and noise.
Dynamics and Control	
Control signal	Learned drift term $u(z, t)$ steering latent trajectories toward observations.
Diffusion coefficient	Scalar or vector g scaling Brownian increments $\mathrm{d}W_t$, injecting uncertainty into latent paths.
Continuous Normalizing Flow (CNF)	Neural ODE providing backbone drift $f_\theta(z, context, t)$; warps latent coordinates while enabling adjoint gradients.
Clamping	Logic that limits control magnitudes and absorbs out-of-manifold trajectories.
Stability	Ensuring numerical and statistical properties remain bounded during integration.
Adjoint method	Technique for computing gradients through ODE integrators by solving backward-in-time adjoint ODEs.
Jacobian	Matrix of partial derivatives; important for drift sensitivity and CNF stability.
Training and Optimization	
Learning rate (LR)	Step size for optimizer updates; affects convergence speed and stability.
Optimizer	Algorithm (e.g., Adam, SGD) updating model parameters using computed gradients.
Training	Phase where model parameters are updated to minimize loss functions like ELBO.
Online/offline	Online processes data in real-time; offline processes pre-collected datasets in batches.
Batching	Dividing data into smaller subsets to improve memory usage and optimization.
Epoch	One complete pass over the training dataset.
Regularization	Loss penalties (e.g., KL, control cost) to prevent overfitting.
Gradient clipping	Limiting gradient magnitude to prevent exploding gradients.
Early stopping	Halting training when validation loss stops improving to avoid overfitting.
Losses and Metrics	
KL divergence	Measures non-symmetric distance between two distributions.
Evidence Lower Bound (ELBO)	Variational objective combining reconstruction, KL, and physics/control penalties.
L_2, L_1 norms	Vector magnitude measures; used in loss functions.
MAE and MSE	Mean Absolute Error (MAE) and Mean Squared Error (MSE); common regression losses.
Overfitting	Model memorizes training noise, failing to generalize.
Physics loss	Loss enforcing consistency with physical laws (e.g., PDE residuals in PINNs).
Residual error	Difference between predicted and true values; often minimized in regression or PINN training.
Neural Network Concepts	
Neural network	Parameterized map of layers and nonlinearities trained via gradient descent.
Activation function	Nonlinear function applied to layer outputs (e.g., ReLU, tanh, GELU).
Time embeddings	Encodes continuous time values into higher-dimensional vectors (Fourier features).
Attention	Mechanism to weigh inputs dynamically based on relevance; used in Transformers.
Transformer	Architecture using self-attention for sequence modeling.
RNN / GRU / LSTM	Recurrent neural networks for temporal data; GRU and LSTM include gates to manage memory.
Inference	Generating predictions or latent trajectories from a trained model.
Batch normalization	Technique normalizing layer inputs to stabilize training.
Dropout	Regularization technique randomly dropping units during training to prevent overfitting.
Physics-Informed Machine Learning	
PINN	Neural network trained to satisfy physical laws described by PDEs/ODEs using physics-based loss.
PDE residual	Difference between predicted solution and PDE evaluated at sample points.
Boundary/Initial conditions	Known values at domain boundaries or initial time guiding PINN training.
Constraint enforcement	Ensuring model predictions respect physical or geometric constraints.
Surrogate modeling	Neural networks approximating expensive simulations while enforcing physics.
Generative Models	
VAE	Variational Autoencoder, probabilistic encoder-decoder learning latent representations.
Diffusion model	Generative model transforming noise into samples via stochastic diffusion.
GAN	Generative Adversarial Network with generator and discriminator trained adversarially.
Flow-based model	Generative model using invertible transformations to map between latent and observed data.
Probabilistic Concepts	
Bayesian inference	Updating beliefs (probability distributions) based on observed data.
Prior	Distribution encoding assumptions before seeing data.
Posterior	Updated distribution after observing data.
Likelihood	Probability of observed data under a given model.
Uncertainty quantification	Estimating confidence or variance in model predictions.
Numerical Methods	
ODE solver	Algorithm for numerically integrating ordinary differential equations (e.g., Runge-Kutta).
Stochastic integration	Integration including random terms (e.g., Brownian motion).
Time discretization	Approximating continuous dynamics with discrete time steps.
Stability analysis	Ensuring numerical methods produce bounded and accurate solutions.
Convergence	Property that solution approaches the true value as step size decreases.
Miscellaneous Concepts	
Numerical stability	Ensuring computations do not diverge or oscillate unphysically.
Surrogate posterior	Learned approximation of the true latent posterior.
Feature embedding	Mapping raw input into higher-dimensional representation for modeling.
Residual connection	Adding input of a layer to its output to improve gradient flow.
Hyperparameter	Configuration parameter (not learned) affecting training and performance.

APPENDIX D
GATED RECURRENT UNIT AND MULTI-LAYER PERCEPTRON

This appendix provides a brief overview of the Gated Recurrent Unit (GRU) and Multi-Layer Perceptron (MLP) architectures used in our model.

A. Neural Networks

Neural networks are computational models inspired by the human brain, consisting of layers of interconnected nodes (neurons) that process input data to learn complex patterns and representations. They are widely used in machine learning tasks such as classification, regression, and generative modeling.

Neural networks are 'trained' using optimization algorithms like gradient descent, which adjust the weights of the connections between neurons to minimize a loss function that quantifies the difference between predicted and true outputs.

Feedforward neural networks, such as MLPs, pass data in one direction from input to output, while recurrent neural networks (RNNs), like GRUs, have connections that form cycles, allowing them to maintain a hidden state and capture temporal dependencies in sequential data.

B. Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a type of feedforward neural network composed of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the subsequent layer, and each connection has an associated weight. The MLP processes input data by performing a series of linear transformations followed by nonlinear activation functions, enabling it to learn complex mappings from inputs to outputs. MLPs are commonly used for tasks such as regression, classification, and function approximation due to their ability to model non-linear relationships.

C. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. This allows RNNs to model temporal dependencies and patterns in sequences, making them suitable for tasks such as language modeling, time series prediction, and speech recognition. However, standard RNNs can suffer from issues like vanishing and exploding gradients, which can hinder their ability to learn long-term dependencies.

D. Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a type of RNN that addresses some of the limitations of standard RNNs by introducing gating mechanisms to control the flow of information. GRUs have two main gates: the update gate and the reset gate. The update gate determines how much of the previous hidden state should be retained, while the reset gate controls how much of the past information to forget. These gates help the GRU maintain long-term dependencies and mitigate the vanishing gradient problem. GRUs are computationally efficient and have been shown to perform well on various sequence modeling tasks, making them a popular choice for applications involving time series and sequential data.

APPENDIX E
HYPERPARAMETER SUMMARY

Table II collects the representative hyperparameters for each flow case we evaluate. Each row highlights the configuration used for the trajectory sampling, CNF backbone training, VSDE controller fitting, and inference diagnostics for that flow.

TABLE II
REPRESENTATIVE SETTINGS FOR EACH FLOW IN THE BENCHMARK SUITE.

Flow	**Key settings (trajectory / CNF / VSDE / inference)**
Euler vortex	4096 particles, 240 steps, $\Delta t = 0.01$ / CNF: batch 512, 8 epochs, lr 0.002, hidden 64, depth 3, context 3, limit 1024 / VSDE: batch 2048, 50 epochs, lr 0.01, 64 particles, 120 steps, control cost 1.0, learnable diffusion / Inference: 64 particles, 120 steps, overlay outputs + phase plots.
Viscous shock tube	1024 particles, 240 steps, $\Delta t = 0.01$ / CNF: batch 512, 8 epochs, lr 0.001, hidden 64, depth 3, context 3, limit 512 / VSDE: batch 128, 4 epochs, lr 0.005, 4 particles, 60 steps, control cost 1.0, learnable diffusion / Inference: 32 particles, 60 steps, overlay outputs + phase plots.
Axial compressor	8192 particles, 180 steps, $\Delta t = 0.005$ / CNF: batch 1024, 10 epochs, lr 0.0015, hidden 128, depth 4, context 4, limit 2048 / VSDE: batch 1024, 30 epochs, lr 0.005, 96 particles, 150 steps, control cost 0.5, learnable diffusion / Inference: 96 particles, 150 steps, overlay outputs + flux diagnostics.

The hyperparameters were selected based on preliminary tuning to balance training stability and inference accuracy across the different flow regimes. The CNF and VSDE settings reflect the complexity of the underlying dynamics, with more challenging flows requiring larger batch sizes, longer training epochs, and finer integration steps to capture the relevant features. The inference configurations ensure sufficient ensemble diversity to evaluate model performance effectively.

You set and view specific hyperparameters in the code repository accompanying this manuscript, which includes configuration files and scripts to reproduce the experimental results reported herein. Those files are located in `src/experiments/`. Inside the files with `.yaml` extensions, you can find the hyperparameter values used for each flow case and experiment step.