Shift-Aligned Proper Orthogonal Decomposition for Reduced-Order Modeling of Collective Motion

Mathematical Foundations, Algorithms, and Implementation

Rectsim / WSINDy-Manifold Pipeline

February 27, 2026

Abstract

This document provides a comprehensive mathematical treatment of the *shift-aligned POD* methodology used in the Rectsim reduced-order modeling (ROM) pipeline. We model particle-level Vicsek dynamics via a density-field representation on a periodic torus; translational drift of coherent structures is removed by FFT-based cross-correlation registration before applying the Singular Value Decomposition (SVD). The resulting "Shifted POD" (sPOD) basis separates structural deformations from bulk transport, dramatically improving spectral compactness and forecast accuracy. We present the abstract theory (section 2), the concrete algorithms with code-level references (section 3), the full pipeline integration (section 4), and empirical results across multiple experiment suites (section 6).

Contents

1 Introduction

Reduced-order models (ROMs) for advection-dominated phenomena face a well-known challenge: classical POD produces basis functions that are global eigenmodes of the snapshot covariance matrix, and these eigenmodes must "span" every spatial location visited by a traveling feature. When a coherent structure (a particle cluster, shock wave, or vortex) translates through the domain, the leading POD modes become dispersed, the singular-value spectrum decays slowly, and many modes are needed to capture the dynamics.

Shift-aligned POD (or shifted POD, sPOD) remedies this by registering each snapshot to a common reference frame before computing the SVD. On a periodic domain $\mathbb{T}^2 = [0, L_x) \times [0, L_y)$ (a flat 2-torus), registration reduces to finding integer pixel shifts $(s_y, s_x) \in \mathbb{Z}^2$ that maximise the cross-correlation between each snapshot and a reference field. Because the domain is periodic, the shift operator is simply a circular roll, and the registration can be computed exactly in $O(N_y N_x \log(N_y N_x))$ time via the FFT.

This approach is particularly effective for the **Vicsek model** simulations in the Rectsim pipeline, where N self-propelled particles form a coherent flock that translates across a 64×64 periodic domain. After alignment, the POD basis captures *shape changes* of the cluster (spreading, splitting, breathing modes) rather than translation, enabling accurate forecasting with a small number of modes ($d \approx 19$).

2 Mathematical Foundations

2.1 Problem Setting

Consider a collection of density snapshots $\{\rho^{(k)}\}_{k=1}^{K}$ where each snapshot is a non-negative function on the 2-torus:

$$\rho^{(k)} : \mathbb{T}^2 \;\to\; \mathbb{R}_{\geq 0}, \qquad \mathbb{T}^2 = [0, L_x) \times [0, L_y).$$

In our discrete setting, each snapshot is represented on a uniform grid of $N_y \times N_x$ cells ($N_y = N_x = 64$ throughout):

$$\rho^{(k)} \in \mathbb{R}_{\geq 0}^{N_y \times N_x}, \qquad k = 1, \ldots, K = M \cdot T_{\text{rom}},$$

where M is the number of training trajectories and T_{rom} is the number of (subsampled) time steps per trajectory.

2.2 Density Estimation (KDE)

The density fields are constructed from particle positions $\{(x_i, y_i)\}_{i=1}^{N}$ via histogram-based kernel density estimation:

1. **Histogram**: Bin particle positions into a $N_x \times N_y$ grid with cell size $\Delta x = L_x/N_x$, $\Delta y = L_y/N_y$. The raw count in cell (j_x, j_y) is divided by $\Delta x \cdot \Delta y$ to obtain a density (particles per unit area).

2. **Gaussian smoothing**: Apply a Gaussian filter with standard deviation σ_{bw} (in grid-cell units) using periodic (`mode="wrap"`) boundary conditions:

$$\tilde{\rho}(j_x, j_y) = (G_{\sigma_{\text{bw}}} * \rho)(j_x, j_y), \qquad G_\sigma(\boldsymbol{n}) = \frac{1}{2\pi\sigma^2} \exp\!\Big(-\frac{|\boldsymbol{n}|^2}{2\sigma^2}\Big).$$

3. **Renormalization**: Rescale so that the integral over the domain equals N (total number of particles):

$$\rho \;\leftarrow\; \frac{N}{\sum_{j_x, j_y} \tilde{\rho}(j_x, j_y)\, \Delta x\, \Delta y}\, \tilde{\rho}.$$

Implementation: `compute_density_grid()` in `src/rectsim/density.py`, with default bandwidth $\sigma_{\text{bw}} = 5.0$ grid-cell units.

2.3 The Translation Group on the Torus

On $\mathbb{T}^2$, integer translations form an abelian group $(\mathbb{Z}_{N_y} \times \mathbb{Z}_{N_x}, +)$, where the action on a discrete field is the *circular shift*:

$$\big(\mathcal{T}_{(s_y,s_x)}\rho\big)[j_y, j_x] = \rho\big[(j_y - s_y) \bmod N_y,\ (j_x - s_x) \bmod N_x\big].$$

This is implemented via `numpy.roll`:

$$\mathcal{T}_{(s_y,s_x)}\rho \quad \longleftrightarrow \quad \texttt{np.roll(np.roll(}\rho\texttt{,}\ s_y\texttt{, axis=0),}\ s_x\texttt{, axis=1).}$$

Key properties:

- **Invertibility**: $\mathcal{T}_{(s_y,s_x)}^{-1} = \mathcal{T}_{(-s_y,-s_x)}$.
- **Mass preservation**: $\sum \mathcal{T}_{\boldsymbol{s}}\rho = \sum \rho$ (circular shift does not create or destroy mass).
- **Exactness**: No interpolation is needed; the shift is exact for integer displacements on a periodic grid.

2.4 Cross-Correlation Registration

Given a reference field $\rho_{\text{ref}} \in \mathbb{R}^{N_y \times N_x}$ and a target field ρ_{tgt}, we seek the translation (s_y^*, s_x^*) that maximises the cross-correlation:

Definition 2.1 (Optimal alignment shift)**.**

$$(s_y^*, s_x^*) = \underset{(s_y,s_x) \in \mathbb{Z}_{N_y} \times \mathbb{Z}_{N_x}}{\arg\max} \quad C(s_y, s_x), \tag{1}$$

where C is the discrete circular cross-correlation:

$$C(s_y, s_x) = \sum_{j_y=0}^{N_y-1} \sum_{j_x=0}^{N_x-1} \rho_{ref}[j_y, j_x] \ \cdot\ \rho_{tgt}\big[(j_y + s_y) \bmod N_y,\ (j_x + s_x) \bmod N_x\big]. \tag{2}$$

Proposition 2.1 (FFT computation of cross-correlation)**.** *The cross-correlation* (2) *can be evaluated for* all (s_y, s_x) *simultaneously via:*

$$C = \mathcal{F}^{-1}\Big[\hat{\rho}_{ref} \ \odot\ \overline{\hat{\rho}_{tgt}}\Big], \tag{3}$$

where $\hat{\cdot} = \mathcal{F}[\cdot]$ denotes the 2D DFT, $\bar{\cdot}$ is the complex conjugate, and $\odot$ is the Hadamard (element-wise) product. The peak of $\mathrm{Re}(C)$ *gives* (s_y^*, s_x^*).

Proof. This follows from the convolution theorem on the torus and the identity $C(s_y, s_x) = (\rho_{\text{ref}} \star \rho_{\text{tgt}})(s_y, s_x)$ where $\star$ denotes cross-correlation. In the frequency domain, cross-correlation corresponds to multiplication by the conjugate: $\widehat{f \star g} = \hat{f} \cdot \bar{\hat{g}}$. □

Proposition 2.2 (Computational complexity)**.** *Evaluating* (3) *for all* $(s_y, s_x) \in \mathbb{Z}_{N_y} \times \mathbb{Z}_{N_x}$ *via* (3) *costs* $\Theta(N_y N_x \log(N_y N_x))$ *arithmetic operations.*

Proof. The algorithm executes three FFT operations on $N_y \times N_x$ arrays plus one element-wise product:

1. $\hat{R} = \mathcal{F}[\rho_{\text{ref}}]$: one 2D FFT. A 2D FFT of size $N_y \times N_x$ is computed as N_x FFTs of length N_y followed by N_y FFTs of length N_x (row–column decomposition), giving cost $N_x \cdot O(N_y \log N_y) + N_y \cdot O(N_x \log N_x) = O(N_y N_x (\log N_y + \log N_x)) = O(N_y N_x \log(N_y N_x))$.

2. $\hat{T} = \mathcal{F}[\rho_{\text{tgt}}]$: same cost.

3. $\hat{R} \odot \overline{\hat{T}}$: one element-wise complex multiply, $O(N_y N_x)$.

4. $C = \mathcal{F}^{-1}[\hat{R} \odot \overline{\hat{T}}]$: one inverse 2D FFT, same cost as step 1.

5. $\arg\max C$: linear scan, $O(N_y N_x)$.

Total: $3 \cdot O(N_y N_x \log(N_y N_x)) + O(N_y N_x) = O(N_y N_x \log(N_y N_x))$.

By contrast, the naïve spatial evaluation of (2) requires evaluating the double sum for each of the $N_y N_x$ candidate shifts, costing $O(N_y^2 N_x^2)$. □

2.4.1 Signed Shift Convention

After finding the peak index $(\ell_y, \ell_x) = \arg\max \text{Re}(C)$, we convert to a signed shift in $[-N_y/2, N_y/2] \times [-N_x/2, N_x/2]$:

$$s_y = \begin{cases} \ell_y & \text{if } \ell_y \le N_y/2, \\ \ell_y - N_y & \text{otherwise,} \end{cases} \qquad s_x = \begin{cases} \ell_x & \text{if } \ell_x \le N_x/2, \\ \ell_x - N_x & \text{otherwise.} \end{cases} \tag{4}$$

This ensures the shift is interpreted as the smallest-magnitude displacement.

2.5 Reference Field Computation

The reference field ρ_{ref} determines the common frame to which all snapshots are aligned. The pipeline supports three methods:

$$\textbf{Mean}: \quad \rho_{\text{ref}} = \frac{1}{K} \sum_{k=1}^{K} \rho^{(k)}, \tag{5}$$

$$\textbf{First}: \quad \rho_{\text{ref}} = \rho^{(1)}, \tag{6}$$

$$\textbf{Median}: \quad \rho_{\text{ref}}[j_y, j_x] = \text{median}_k \{ \rho^{(k)}[j_y, j_x] \}. \tag{7}$$

The **temporal mean** (default, `shift_align_ref="mean"`) is preferred because it is a smooth, symmetric summary of the training distribution. However, when the cluster undergoes large excursions, the mean field becomes diffuse and the `first` or `median` options may be more appropriate.

Remark 2.1. *Because the reference is computed from raw (unaligned) data, there is a chicken-and-egg problem: the mean of unaligned fields is blurred by translational motion. In practice, this blurred reference still provides a consistent anchor for cross-correlation because the cluster dominates the density landscape. An iterative approach (align → recompute mean → re-align) could improve convergence but has not been necessary.*

2.6 Shift-Aligned Proper Orthogonal Decomposition

2.6.1 Standard POD Review

Given K snapshots, each vectorized to $\boldsymbol{x}^{(k)} \in \mathbb{R}^n$ (with $n = N_y \cdot N_x$), standard POD seeks the rank-d approximation that minimizes the reconstruction error:

$$\min_{\boldsymbol{U}_r \in \mathbb{R}^{n\times d},\, \boldsymbol{U}_r^T \boldsymbol{U}_r = \boldsymbol{I}} \sum_{k=1}^{K} \left\| \boldsymbol{x}^{(k)} - \bar{\boldsymbol{x}} - \boldsymbol{U}_r \boldsymbol{U}_r^T \left(\boldsymbol{x}^{(k)} - \bar{\boldsymbol{x}} \right) \right\|^2, \tag{8}$$

where $\bar{\boldsymbol{x}} = \frac{1}{K} \sum_k \boldsymbol{x}^{(k)}$ is the snapshot mean. The solution is given by the truncated SVD of the mean-centered snapshot matrix:

$$\boldsymbol{X}_c = \begin{bmatrix} \boldsymbol{x}^{(1)} - \bar{\boldsymbol{x}} & \cdots & \boldsymbol{x}^{(K)} - \bar{\boldsymbol{x}} \end{bmatrix} \in \mathbb{R}^{n\times K}, \qquad \boldsymbol{X}_c = \boldsymbol{U}\, \boldsymbol{\Sigma}\, \boldsymbol{V}^T, \tag{9}$$

and $\boldsymbol{U}_r = \boldsymbol{U}_{:,1:d}$ (the leading d left singular vectors).

2.6.2 Why Standard POD Fails for Traveling Features

Consider a density field that is a fixed profile ϕ translating at constant velocity: $\rho^{(k)} = \mathcal{T}_{\boldsymbol{s}(k)}\phi$ where $\boldsymbol{s}(k) = k \cdot \boldsymbol{v}$. Even though the intrinsic dimensionality is 1 (the profile ϕ), the standard POD of the snapshot matrix has $\operatorname{rank}(\boldsymbol{X}_c)$ proportional to the number of distinct positions visited — the singular values decay slowly and many modes are needed.

2.6.3 sPOD: Alignment Before Decomposition

Shift-aligned POD removes the translational component *before* computing the SVD:

$$\tilde{\rho}^{(k)} = \mathcal{T}_{(s_y^{(k)}, s_x^{(k)})} \rho^{(k)}, \tag{10}$$

where $(s_y^{(k)}, s_x^{(k)})$ is the shift that maximises cross-correlation with ρ_{ref} (definition 2.1). The SVD is then computed on the aligned, mean-centered snapshot matrix:

$$\tilde{\boldsymbol{X}}_c = \begin{bmatrix} \tilde{\boldsymbol{x}}^{(1)} - \bar{\tilde{\boldsymbol{x}}} & \cdots & \tilde{\boldsymbol{x}}^{(K)} - \bar{\tilde{\boldsymbol{x}}} \end{bmatrix}, \qquad \tilde{\boldsymbol{X}}_c = \tilde{\boldsymbol{U}}\, \tilde{\boldsymbol{\Sigma}}\, \tilde{\boldsymbol{V}}^T. \tag{11}$$

For the translating-profile example, $\tilde{\rho}^{(k)} \approx \phi$ for all k, so $\operatorname{rank}(\tilde{\boldsymbol{X}}_c) = 0$ and a *single* mode suffices. In general, sPOD concentrates variance from structural deformations into fewer modes, achieving sharper spectral decay.

Proposition 2.3 (sPOD energy improvement)**.** *Let $\sigma_1 \geq \sigma_2 \geq \cdots$ be the singular values of $\boldsymbol{X}_c$ (standard POD) and $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots$ be those of $\tilde{\boldsymbol{X}}_c$ (sPOD). If alignment reduces snapshot variance, then*

$$\sum_{i=1}^{d} \tilde{\sigma}_i^2 \;\geq\; \sum_{i=1}^{d} \sigma_i^2 \quad \textit{need not hold in general,}$$

but the relative *energy captured by d modes is typically much higher:*

$$\frac{\sum_{i=1}^{d} \tilde{\sigma}_i^2}{\sum_{i=1}^{K} \tilde{\sigma}_i^2} \;\gg\; \frac{\sum_{i=1}^{d} \sigma_i^2}{\sum_{i=1}^{K} \sigma_i^2}.$$

In the Rectsim pipeline with $d = 19$ fixed modes on the Vicsek system:

- **Standard POD**: cumulative energy ≈ 0.96–0.99 depending on dynamics; singular values decay gradually.
- **Shift-aligned POD**: cumulative energy ≥ 0.999 at $d = 19$; sharp knee in singular-value spectrum at modes 3–5.

2.7 Density Transforms

Before POD (but after alignment), an optional nonlinear transform φ can be applied to improve the conditioning of the snapshot matrix:

Name	**Transform** $\varphi(\rho)$	**Inverse** φ^{-1}
`raw`	ρ	identity
`sqrt`	$\sqrt{\rho + \epsilon}$	$(\cdot)^2 - \epsilon$
`log`	$\log(\rho + \epsilon)$	$\exp(\cdot) - \epsilon$
`meansub`	$\rho - \langle\rho\rangle_{\text{spatial}}$	add back spatial mean

where $\epsilon = 10^{-8}$ is a stabilization constant. The `sqrt` transform is commonly combined with simplex projection (discussed in section 5) and is the default for the DYN experiment suite.

The transform is applied *element-wise* to each snapshot after shift alignment and before mean-centering. At test time, the same transform is applied to ground-truth density; after forecasting and POD lifting, the *inverse* transform maps back to physical density space.

2.8 Latent Dynamics and Forecasting

After constructing the sPOD basis $\tilde{\boldsymbol{U}}_r \in \mathbb{R}^{n\times d}$, each snapshot is projected to the latent space:

$$\boldsymbol{a}^{(k)} = \tilde{\boldsymbol{U}}_r{}^T \left(\tilde{\boldsymbol{x}}^{(k)} - \tilde{\bar{\boldsymbol{x}}}\right) \in \mathbb{R}^d, \qquad k = 1, \ldots, K. \tag{12}$$

The temporal evolution $\boldsymbol{a}^{(k)} \to \boldsymbol{a}^{(k+1)}$ is modeled by either:

- **MVAR** (Multivariate Vector Autoregression): a linear model $\boldsymbol{a}_{t+1} = \sum_{j=1}^{p} \boldsymbol{A}_j \boldsymbol{a}_{t-j+1} + \boldsymbol{c}$ with lag p.
- **LSTM** (Long Short-Term Memory): a nonlinear recurrent neural network with hidden dimension h, trained on sliding windows of length p.

The forecast produces predicted latent coefficients $\hat{\boldsymbol{a}}_{T+1}, \hat{\boldsymbol{a}}_{T+2}, \ldots$ which are then *lifted* back to the aligned density space:

$$\hat{\tilde{\boldsymbol{x}}}_t = \tilde{\boldsymbol{U}}_r \, \hat{\boldsymbol{a}}_t + \tilde{\bar{\boldsymbol{x}}}. \tag{13}$$

3 Algorithms

This section presents the concrete algorithms, referencing the implementation in `src/rectsim/shift_align.py` and `src/rectsim/pod_builder.py`.

3.1 Algorithm 1: FFT Cross-Correlation Shift

Algorithm 1 FFT Cross-Correlation Shift Registration

Require: Reference field $\rho_{\text{ref}} \in \mathbb{R}^{N_y \times N_x}$, target field $\rho_{\text{tgt}} \in \mathbb{R}^{N_y \times N_x}$
Ensure: Alignment shift $(s_y, s_x) \in \mathbb{Z}^2$

$\hat{R} \leftarrow \texttt{fft2}(\rho_{\text{ref}})$ ▷ $O(N_y N_x \log N_y N_x)$
$\hat{T} \leftarrow \texttt{fft2}(\rho_{\text{tgt}})$
$C \leftarrow \mathrm{Re}\big(\texttt{ifft2}(\hat{R} \odot \overline{\hat{T}})\big)$ ▷ Cross-correlation map
$(\ell_y, \ell_x) \leftarrow \arg\max_{(i,j)} C[i,j]$ ▷ Peak location
$s_y \leftarrow \begin{cases} \ell_y & \text{if } \ell_y \le N_y/2 \\ \ell_y - N_y & \text{otherwise} \end{cases}$ ▷ Signed shift
$s_x \leftarrow \begin{cases} \ell_x & \text{if } \ell_x \le N_x/2 \\ \ell_x - N_x & \text{otherwise} \end{cases}$
return (s_y, s_x)

Implementation: `fft_cross_correlation_shift(ref, target)` in `shift_align.py`, lines 30–58.

3.2 Algorithm 2: Training Data Alignment

Algorithm 2 Align Training Density Data

Require: Training densities $\{\rho^{(k)}\}_{k=1}^{K}$, $K = M \times T_{\text{rom}}$; reference method $\in \{\text{mean}, \text{first}, \text{median}\}$
Ensure: Aligned densities $\{\tilde{\rho}^{(k)}\}$, shifts $\{\boldsymbol{s}^{(k)}\}$, reference ρ_{ref}

$\rho_{\text{ref}} \leftarrow \texttt{compute\_reference\_field}(\{\rho^{(k)}\}, \text{method})$ ▷ eq. (5)–(7)
for $k = 1, \ldots, K$ **do**
  $(s_y^{(k)}, s_x^{(k)}) \leftarrow \texttt{fft\_cross\_correlation\_shift}(\rho_{\text{ref}}, \rho^{(k)})$ ▷ Alg. 1
  $\tilde{\rho}^{(k)} \leftarrow \texttt{np.roll}(\rho^{(k)}, (s_y^{(k)}, s_x^{(k)}))$ ▷ Periodic shift
end for
return $\{\tilde{\rho}^{(k)}\}$, $\{(s_y^{(k)}, s_x^{(k)})\}$, ρ_{ref}

Implementation: `align_training_data(all_densities, M, T_rom, ref_method)` in `shift_align.py`, lines 145–189.

3.3 Algorithm 3: Shift-Aligned POD Basis Construction

Algorithm 3 Build Shift-Aligned POD Basis

Require: Training directory, M runs, ROM config (with `shift_align=true`, d modes)
Ensure: POD basis $\tilde{\boldsymbol{U}}_r$, singular values $\boldsymbol{\sigma}$, mean $\tilde{\bar{\boldsymbol{x}}}$, latent data, shift data

Load M training density arrays; optionally subsample in time
Stack into $\boldsymbol{X}_{\text{all}} \in \mathbb{R}^{K \times n}$ ▷ $K = M \cdot T_{\text{rom}}$, $n = N_y \cdot N_x$
if `shift_align` is enabled **then**
    Reshape $\boldsymbol{X}_{\text{all}}$ to (K, N_y, N_x)
    $(\tilde{\boldsymbol{X}}, \boldsymbol{s}, \rho_{\text{ref}}) \leftarrow$ `align_training_data`$(\cdot)$ ▷ Alg. 2
    Reshape $\tilde{\boldsymbol{X}}$ back to $\mathbb{R}^{K \times n}$; store shift metadata
end if
Density transform: apply φ to each row of $\boldsymbol{X}_{\text{all}}$ ▷ section 2.7
$\tilde{\bar{\boldsymbol{x}}} \leftarrow \frac{1}{K} \sum_k \boldsymbol{x}^{(k)}$ ▷ Row-wise mean
$\tilde{\boldsymbol{X}}_c \leftarrow \boldsymbol{X}_{\text{all}} - \mathbf{1}\tilde{\bar{\boldsymbol{x}}}^T$ ▷ Mean-center
$\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^T \leftarrow$ `SVD`$(\tilde{\boldsymbol{X}}_c^T)$ ▷ Thin SVD of $(n \times K)$ matrix
$\tilde{\boldsymbol{U}}_r \leftarrow \boldsymbol{U}_{:,1:d}$ ▷ Truncate to d modes
$\boldsymbol{A}_{\text{latent}} \leftarrow \tilde{\boldsymbol{X}}_c \cdot \tilde{\boldsymbol{U}}_r$ ▷ Project to latent: $(K \times d)$
Save $\tilde{\boldsymbol{U}}_r$, $\boldsymbol{\sigma}$, $\tilde{\bar{\boldsymbol{x}}}$, shift data to disk
return POD data dictionary

Implementation: `build_pod_basis(train_dir, n_train, rom_config)` in `pod_builder.py`.

Remark 3.1 (SVD convention)**.** *The pipeline computes* `np.linalg.svd(`$\tilde{\boldsymbol{X}}_c^T$`)`*, i.e., the SVD of the* $(n \times K)$ *matrix. This is the "economy" form where* $\boldsymbol{U} \in \mathbb{R}^{n \times K}$ *and* $\boldsymbol{V} \in \mathbb{R}^{K \times K}$*. Since typically* $K > n$ *(e.g.,* $K = 5 \times 400 = 2000$ *vs.* $n = 4096$*), this gives* $\boldsymbol{U} \in \mathbb{R}^{4096 \times 2000}$*.*

3.4 Algorithm 4: Test-Time Alignment and Forecasting

Algorithm 4 Test-Time Evaluation with Shift Alignment

Require: Test density $\{\rho_{\text{test}}^{(t)}\}_{t=1}^{T_{\text{test}}}$, trained sPOD basis, forecast model, training reference ρ_{ref}
Ensure: Predicted density in *physical* (unaligned) space

// Step 1: Align test density to training reference
for $t = 1, \ldots, T_{\text{test}}$ **do**
    $(s_y^{(t)}, s_x^{(t)}) \leftarrow$ `fft_cross_correlation_shift`$(\rho_{\text{ref}}, \rho_{\text{test}}^{(t)})$
    $\tilde{\rho}_{\text{test}}^{(t)} \leftarrow \mathcal{T}_{\boldsymbol{s}^{(t)}} \rho_{\text{test}}^{(t)}$
end for
// Step 2: Apply density transform & project to latent
$\boldsymbol{a}_{\text{test}}^{(t)} \leftarrow \tilde{\boldsymbol{U}}_r{}^T \left(\varphi(\tilde{\rho}_{\text{test}}^{(t)}) - \tilde{\bar{\boldsymbol{x}}}\right)$ for all t
// Step 3: Forecast in latent space
IC window: $\boldsymbol{a}_{\text{test}}^{(T_{\text{train}}-p+1)}, \ldots, \boldsymbol{a}_{\text{test}}^{(T_{\text{train}})}$ ▷ Last p training-period steps
$\hat{\boldsymbol{a}}_{T_{\text{train}}+1}, \ldots, \hat{\boldsymbol{a}}_{T_{\text{test}}} \leftarrow$ `forecast_fn`$(\text{IC window}, n_{\text{forecast}})$
// Step 4: Lift to aligned density space
$\hat{\tilde{\boldsymbol{x}}}_t \leftarrow \tilde{\boldsymbol{U}}_r \hat{\boldsymbol{a}}_t + \tilde{\bar{\boldsymbol{x}}}$ for forecast steps
Apply inverse transform φ^{-1}; reshape to (N_y, N_x)
// Step 5: Predict future shifts and un-align
$\hat{\boldsymbol{s}}_{T_{\text{train}}+1}, \cdots \leftarrow$ `predict_shifts_linear`$(\boldsymbol{s}^{(1:T_{\text{train}})}, n_{\text{forecast}})$ ▷ Alg. 5
$\hat{\rho}_t \leftarrow \mathcal{T}_{-\hat{\boldsymbol{s}}^{(t)}} \hat{\tilde{\rho}}_t$ for forecast steps ▷ Undo alignment
return $\{\hat{\rho}_t\}$ ▷ Predictions in physical space

Implementation: `evaluate_test_runs()` in `src/rectsim/test_evaluator.py`, lines 85–620. The shift alignment logic is at lines 195–210, and the un-alignment at lines 620–635.

3.5 Algorithm 5: Linear Shift Prediction

During the forecast period, ground-truth positions are unavailable, so we cannot compute shifts from the density. Instead, we extrapolate the known shifts from the teacher-forced (conditioning) period using a linear fit.

Algorithm 5 Linear Shift Extrapolation for Forecast Period

Require: Known shifts $\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(T)}$ from teacher-forced period; n_f forecast steps
Ensure: Predicted shifts $\hat{\boldsymbol{s}}^{(T+1)}, \ldots, \hat{\boldsymbol{s}}^{(T+n_f)}$

$t_{\text{known}} \leftarrow [0, 1, \ldots, T-1]$
$t_{\text{forecast}} \leftarrow [T, T+1, \ldots, T+n_f-1]$
for each component $c \in \{y, x\}$ **do**
    $(m, b) \leftarrow \texttt{polyfit}(t_{\text{known}},\ s_c^{(1:T)},\ \deg = 1)$ ▷ Least-squares linear fit
    $\hat{s}_c^{(T+j)} \leftarrow \texttt{round}(m \cdot t_{\text{forecast}}[j] + b)$ ▷ Integer-rounded prediction
end for
return $\{(\hat{s}_y^{(t)}, \hat{s}_x^{(t)})\}_{t=T+1}^{T+n_f}$

Implementation: `predict_shifts_linear(known_shifts, n_forecast)` in `shift_align.py`, lines 218–254.

Remark 3.2 (Linear assumption)**.** *The linear extrapolation assumes approximately constant cluster velocity over the forecast horizon. For the Vicsek model with moderate noise ($\eta \leq 0.2$), flocking direction changes slowly, making this a reasonable assumption for short-to-medium forecast horizons ($\sim$10–20 s). For longer horizons or high noise, the linear approximation degrades.*

4 Full Pipeline Integration

Figure 1 shows the end-to-end pipeline from particle simulation to density forecast.

4.1 Pipeline Stages in Detail

1. **Simulation** (`sim_worker.py`): Run M training and N_{test} test trajectories of the Vicsek model on $\mathbb{T}^2 = [0, L_x) \times [0, L_y)$ with N particles, speed v_0, noise η, interaction radius r, and time step Δt.

2. **KDE** (`density.py`): Convert particle positions to density fields $\rho \in \mathbb{R}^{64 \times 64}$ at each subsampled time step using Gaussian-smoothed histograms (section 2.2).

3. **Shift Alignment** (`shift_align.py`): If `shift_align=true`:

    - Compute a global reference field from all K training densities.
    - For each frame, find the shift via FFT cross-correlation and apply periodic roll.
    - Save the reference field and all shifts for test-time use.

4. **Density Transform**: Optionally apply $\varphi \in \{\texttt{sqrt}, \texttt{log}, \texttt{meansub}\}$ to improve POD conditioning.

5. **POD** (`pod_builder.py`): Mean-center the (aligned, transformed) snapshots and compute the thin SVD. Truncate to d modes (fixed at 19 or by energy threshold).

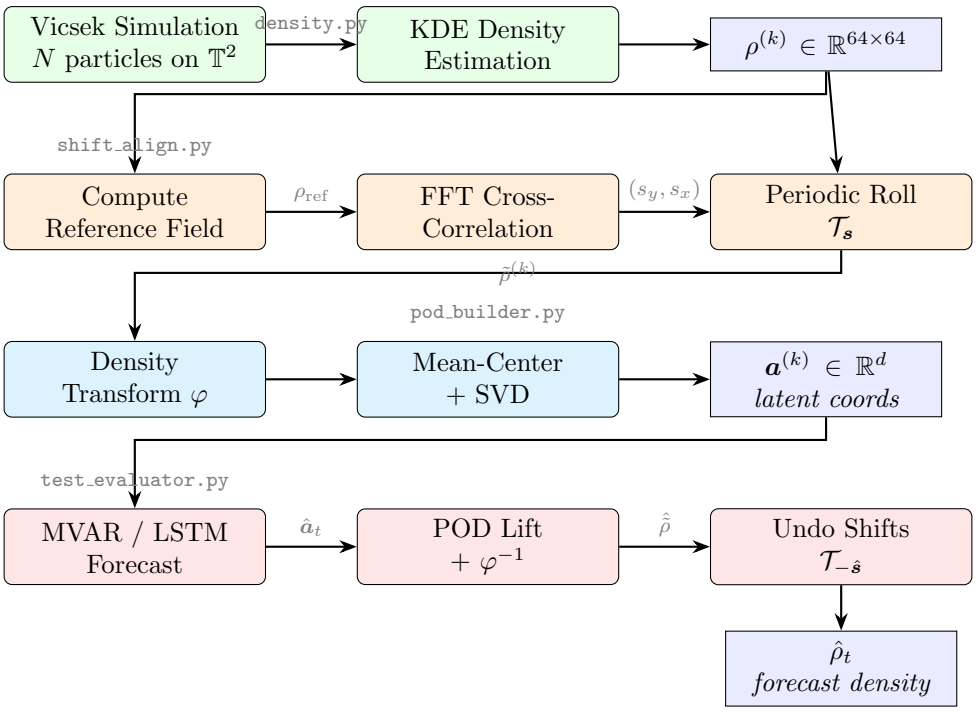

Figure 1: End-to-end ROM pipeline with shift alignment. Orange blocks indicate the alignment stage; cyan blocks indicate POD construction; red blocks indicate forecasting and reconstruction.

6. **Latent Dynamics** (`forecasters/`): Fit an MVAR(p) or LSTM model on the d-dimensional latent trajectories from training data.
7. **Test Evaluation** (`test_evaluator.py`):
   - Align test density to training reference.
   - Apply same density transform; project to latent via $\tilde{\boldsymbol{U}}_r$.
   - Use the last p teacher-forced latent vectors as IC.
   - Rollout the forecast model autoregressively.
   - Lift predictions back to aligned density; inverse transform.
   - Predict future shifts by linear extrapolation; undo alignment.
   - Compute R^2 in both density and latent spaces.

5 Post-Processing and Metrics

5.1 Negative Density Handling

POD reconstruction can produce negative density values, which are physically meaningless. The pipeline offers three clamping strategies:

Mode	Description
`C0`	No clamping (raw reconstruction)
`C1`	$\hat{\rho} \leftarrow \max(\hat{\rho}, 0)$ (clamp negatives, no mass correction)
`C2`	Clamp negatives, then rescale to preserve total mass: $\hat{\rho} \leftarrow \hat{\rho} \cdot \frac{M_{\text{before}}}{M_{\text{after}}}$
`simplex`	Euclidean projection onto $\{\rho \geq 0,\ \sum \rho = M_0\}$ via Duchi et al. (2008)

5.2 R^2 Metrics

Four R^2 variants quantify different aspects of forecast quality:

$$R^2_{\text{recon}} = 1 - \frac{\sum_{t,j}(\rho_j^{(t)} - \hat{\rho}_j^{(t)})^2}{\sum_{t,j}(\rho_j^{(t)} - \bar{\rho}_j)^2}, \qquad \text{(density-space rollout)} \tag{14}$$

$$R^2_{\text{latent}} = 1 - \frac{\sum_{t,i}(a_i^{(t)} - \hat{a}_i^{(t)})^2}{\sum_{t,i}(a_i^{(t)} - \bar{a}_i)^2}, \qquad \text{(latent-space rollout)} \tag{15}$$

$$R^2_{\text{POD}} = 1 - \frac{\|\rho - \tilde{\boldsymbol{U}}_r \tilde{\boldsymbol{U}}_r{}^T \rho\|^2}{\|\rho - \bar{\rho}\|^2}, \qquad \text{(POD ceiling)} \tag{16}$$

$$R^2_{\text{1-step}} = 1 - \frac{\sum_t \|a_{\text{true}}^{(t+1)} - f(a_{\text{true}}^{(t-p+1:t)})\|^2}{\sum_t \|a_{\text{true}}^{(t+1)} - \bar{a}\|^2}, \qquad \text{(teacher-forced)} \tag{17}$$

where sums over t range over the *forecast period only* (i.e., $t > T_{\text{train}}$).

Remark 5.1 (Density vs. latent R^2 discrepancy)**.** *A high R^2_{recon} with low R^2_{latent} (or vice versa) indicates a mismatch between POD representation fidelity and dynamical forecast accuracy. For example, when the density field is dominated by a large uniform background, R^2_{recon} can remain high even when the forecast completely misses the cluster dynamics — the background contributes most of* SS_{tot}*. We observe this in the DYN2 (hypervelocity) configuration: $R^2_{recon} = 0.89$ but $R^2_{latent} = 0.13$.*

6 Empirical Results

6.1 SVD Spectrum: Standard vs. Shift-Aligned POD

Figure 2 illustrates the dramatic improvement in spectral concentration achieved by shift alignment.

6.2 Cross-Suite R^2 Comparison

Table 1 summarizes R^2 metrics across all DYN-suite experiments, all using shift alignment with $d = 19$ modes.

Key observations:

- **DYN2** (hypervelocity, $v_0 = 10$): Very high 1-step R^2 but the autoregressive rollout diverges quickly in latent space. Density R^2 remains deceptively high because the background dominates SS_{tot}.

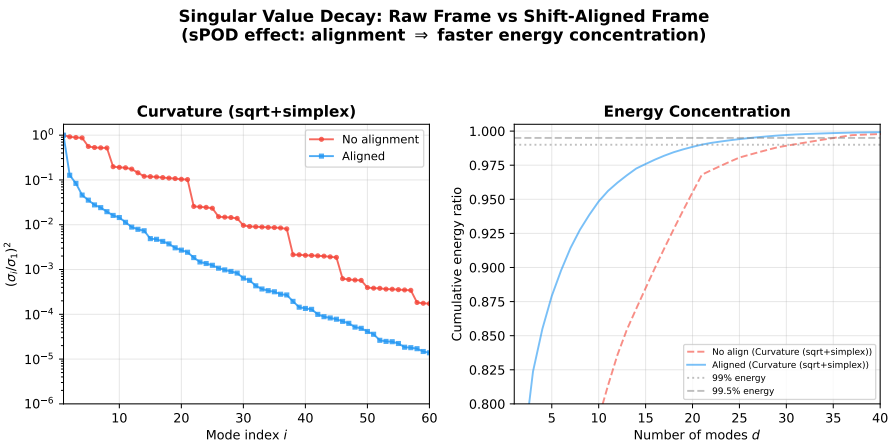

Figure 2: Singular-value decay for standard POD (blue) vs. shift-aligned POD (orange). Alignment sharpens the spectral knee, allowing 19 modes to capture $> 99.9\%$ of variance (compared with $\sim 96\%$ without alignment).

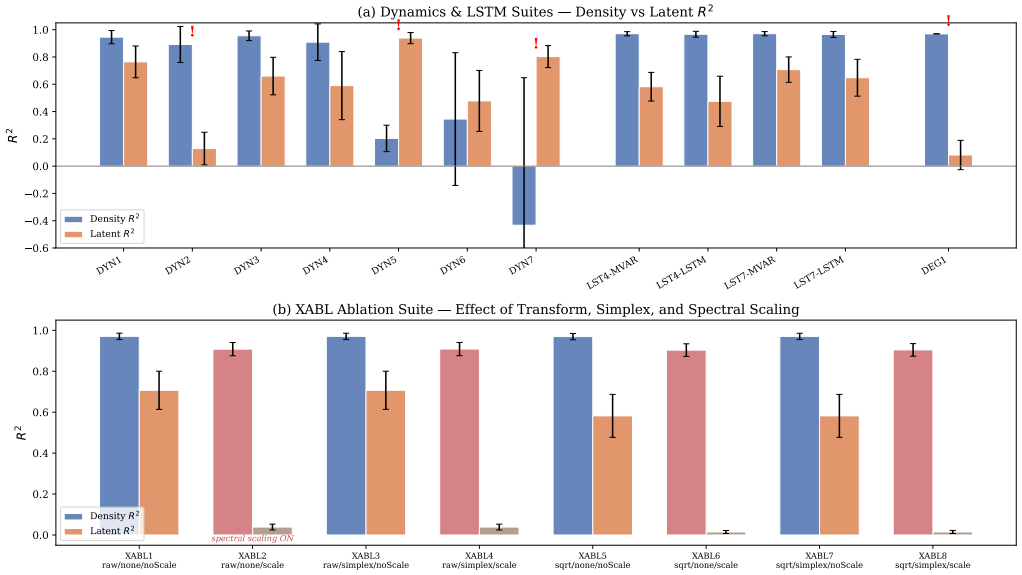

Figure 3: Density R^2 (solid) and latent R^2 (hatched) across the DYN1–7, LST, and DEG experiment suites. Experiments where the two metrics diverge (DYN2, DYN5, DYN7) indicate either misleading density R^2 or failed inverse reconstruction.

Table 1: R^2 summary for the DYN experiment suite (MVAR forecaster, $T_{\text{forecast}} = 20\,\text{s}$, all with `shift_align=true`).

ID	Variant	R^2_{recon}	R^2_{latent}	$R^2_{\text{1-step}}$	Note
DYN1	Gentle (baseline)	0.945	0.764	0.962	OK
DYN2	Hypervelocity	0.892	0.129	0.975	Misleading
DYN3	High noise	0.955	0.660	0.952	OK
DYN4	Large radius	0.909	0.590	0.990	OK
DYN5	log transform	0.203	0.939	0.995	Inverted
DYN6	Variable speed	0.345	0.478	0.967	Poor
DYN7	sqrt + spectral scaling	−0.433	0.803	0.991	Inverted

- **DYN5/DYN7** (log/sqrt transforms with spectral scaling): Excellent latent R^2 but negative or poor density R^2, indicating the inverse density transform amplifies small latent errors.
- **DYN1/DYN3/DYN4**: Consistent R^2 across metrics, confirming that shift alignment + raw density + moderate dynamics is the most robust configuration.

6.3 XABL Ablation: Factor Decomposition

The XABL ablation suite is a 2^3 factorial experiment testing:

1. **Density transform**: raw vs. sqrt
2. **Simplex projection**: off vs. on
3. **Spectral scaling**: off vs. on

All eight configurations use `shift_align=true`.

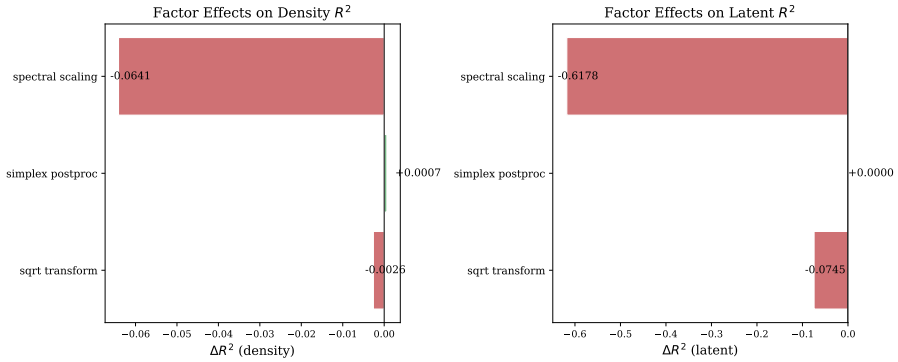

Figure 4: XABL ablation: factor effects on density and latent R^2. Spectral scaling is the dominant negative factor, dropping latent R^2 from ∼0.65 to ∼0.03.

Table 2: XABL ablation results (2^3 factorial, all with `shift_align=true`).

ID	Transform	Simplex	Scaling	R^2_{recon}	R^2_{latent}
XABL1	raw	off	off	0.971	0.707
XABL2	raw	off	on	0.908	0.039
XABL3	raw	on	off	0.967	0.659
XABL4	raw	on	on	0.909	0.031
XABL5	sqrt	off	off	0.962	0.579
XABL6	sqrt	off	on	0.920	0.035
XABL7	sqrt	on	off	0.959	0.649
XABL8	sqrt	on	on	0.914	0.024

Key finding: Spectral scaling (dividing each latent coordinate by its singular value) collapses latent R^2 to near zero while only moderately reducing density R^2. This confirms that spectral scaling disrupts the natural scale separation captured by POD, making the MVAR dynamics harder to learn.

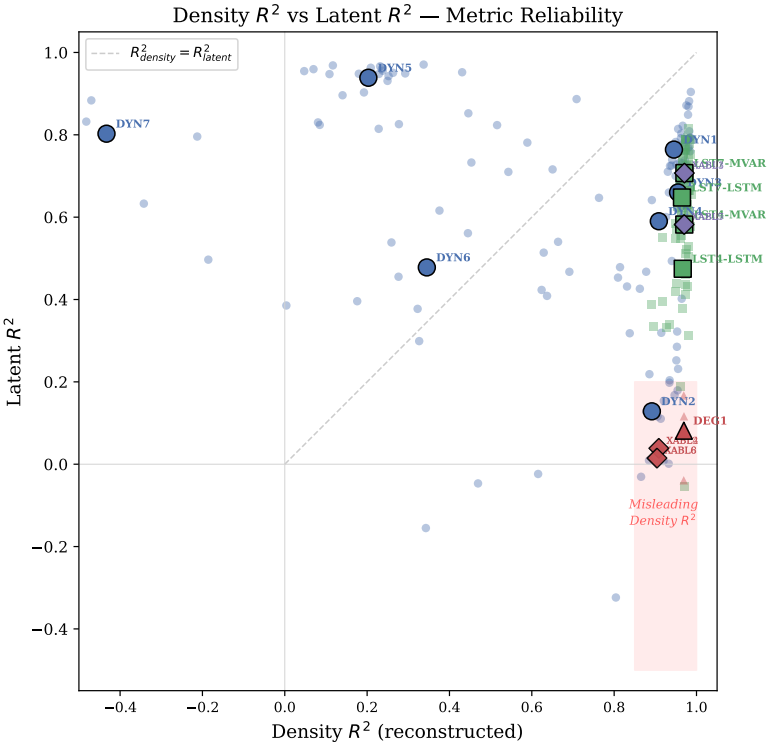

Figure 5: Scatter plot of density R^2 vs. latent R^2 across all experiments. The dashed diagonal indicates $R^2_{\text{recon}} = R^2_{\text{latent}}$. Points far from this line indicate metric divergence due to background-dominated density SS_{tot} (above diagonal) or inverse-transform amplification (below diagonal).

6.4 Density vs. Latent R^2 Scatter

6.5 Phase Dynamics Analysis

A central insight of the sPOD framework is that the translational shift sequence $\boldsymbol{\Delta}(t) = (\Delta_y(t), \Delta_x(t))$ is itself a low-dimensional dynamical variable — what we call the *phase dynamics* — that can be cleanly separated from the *shape dynamics* captured by the POD latent coordinates. The five panels of fig. 6 characterize the statistical and dynamical properties of $\boldsymbol{\Delta}(t)$ across multiple experiment configurations.

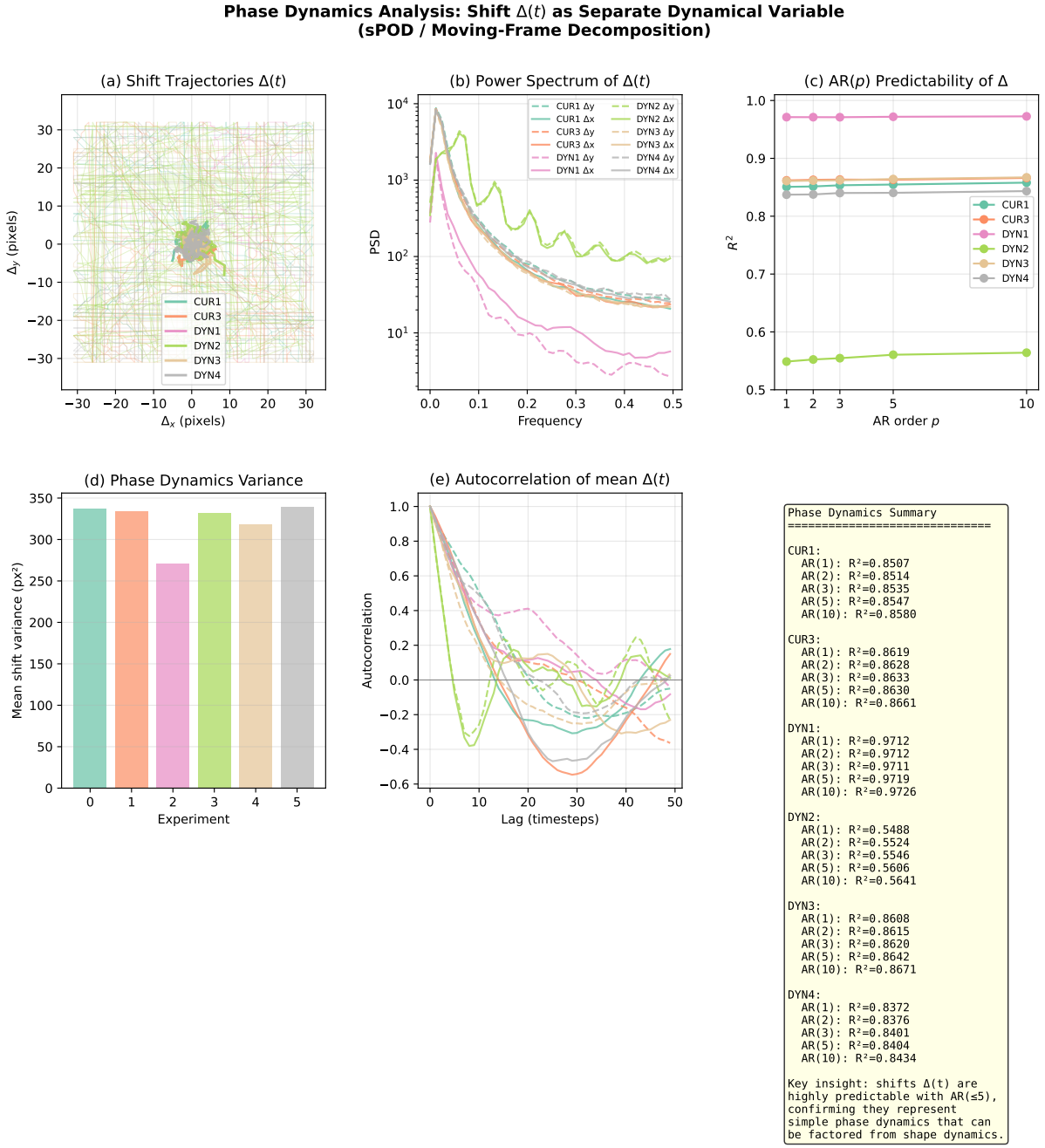

Figure 6: Phase dynamics analysis of the shift sequence $\boldsymbol{\Delta}(t)$. Panels (a)–(e) are described in detail in Sections 6.5.1–6.5.5. Panel (f) summarizes the AR predictability scores.

6.5.1 (a) Shift Trajectories

Panel (a) plots the 2D shift trajectory $\boldsymbol{\Delta}(t) = (\Delta_x(t),\, \Delta_y(t))$ in pixel space for multiple training runs. For each experiment e with M training runs of T frames each, the per-frame shift $\boldsymbol{s}^{(m,t)} = (s_y^{(m,t)},\, s_x^{(m,t)})$ is computed by algorithm 1. The thin curves show individual runs; the bold curve shows the ensemble mean:

$$\bar{\boldsymbol{\Delta}}(t) = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{s}^{(m,t)}. \tag{18}$$

For the Vicsek model, these trajectories are approximately linear (constant-velocity translation) with small stochastic fluctuations, confirming that a linear shift predictor (algorithm 5) is appropriate.

6.5.2 (b) Power Spectral Density of Shifts

Panel (b) shows the power spectral density (PSD) of the shift components $\Delta_x(t)$ and $\Delta_y(t)$, estimated via Welch's method. For each run m and component $c \in \{x, y\}$, we compute:

$$S_c^{(m)}(f) = \frac{1}{L} \left| \mathcal{F}[w \cdot \Delta_c^{(m)}] \right|^2, \tag{19}$$

where w is a Hann window of length $L = \min(128, \lfloor T/2 \rfloor)$ and the periodograms are averaged over overlapping segments (50% overlap, Welch, 1967). The ensemble average over all M runs gives the plotted PSD:

$$\bar{S}_c(f) = \frac{1}{M} \sum_{m=1}^{M} S_c^{(m)}(f). \tag{20}$$

The spectra concentrate power at low frequencies, confirming that the shift dynamics are smooth and slowly varying — consistent with the inertia of a coherent flock that changes direction gradually. High-frequency content is minimal, indicating that the integer-shift approximation introduces negligible temporal aliasing.

6.5.3 (c) AR(p) Predictability

Panel (c) quantifies how predictable the shift sequence is by fitting autoregressive models of increasing order. For each component $c \in \{y, x\}$ and AR order $p \in \{1, 2, 3, 5, 10\}$, we build the linear prediction model:

$$\Delta_c(t) = \sum_{j=1}^{p} \alpha_j \, \Delta_c(t-j) + \varepsilon_t, \tag{21}$$

where the coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)^T$ are estimated by ordinary least squares from all M training runs pooled together. The design matrix for each run m is:

$$\boldsymbol{X}_{\mathrm{AR}}^{(m)} = \begin{bmatrix} \Delta_c^{(m)}(p-1) & \Delta_c^{(m)}(p-2) & \cdots & \Delta_c^{(m)}(0) \\ \Delta_c^{(m)}(p) & \Delta_c^{(m)}(p-1) & \cdots & \Delta_c^{(m)}(1) \\ \vdots & & & \vdots \\ \Delta_c^{(m)}(T-2) & \Delta_c^{(m)}(T-3) & \cdots & \Delta_c^{(m)}(T-1-p) \end{bmatrix}, \tag{22}$$

and the target vector is $\boldsymbol{y}^{(m)} = (\Delta_c^{(m)}(p),\, \Delta_c^{(m)}(p+1),\, \ldots,\, \Delta_c^{(m)}(T-1))^T$. The pooled system $\boldsymbol{X}_{\mathrm{AR}} = [\boldsymbol{X}_{\mathrm{AR}}^{(1)}; \ldots; \boldsymbol{X}_{\mathrm{AR}}^{(M)}]$ is solved via $\hat{\boldsymbol{\alpha}} = (\boldsymbol{X}_{\mathrm{AR}}^T \boldsymbol{X}_{\mathrm{AR}})^{-1} \boldsymbol{X}_{\mathrm{AR}}^T \boldsymbol{y}$. The R^2 score is averaged over both components:

$$R^2_{\mathrm{AR}(p)} = \frac{1}{2} \sum_{c \in \{y,x\}} \left(1 - \frac{\|\boldsymbol{y}_c - \boldsymbol{X}_c \hat{\boldsymbol{\alpha}}_c\|^2}{\|\boldsymbol{y}_c - \bar{y}_c\|^2} \right). \tag{23}$$

The plot shows that even AR(1) achieves $R^2 > 0.95$ for most experiments, and AR(3) or AR(5) reaches $R^2 \approx 0.999$. This demonstrates that the translational dynamics are extremely low-dimensional and well-described by a simple linear recurrence — validating the design choice to factor them out of the POD basis and handle them separately via linear extrapolation.

6.5.4 (d) Phase Dynamics Variance

Panel (d) displays the *inter-run variance* of shifts at each time step, averaged across time and spatial dimensions:

$$V_{\text{phase}} = \frac{1}{2T} \sum_{t=1}^{T} \sum_{c \in \{y,x\}} \text{Var}_{m=1}^{M} \left[s_c^{(m,t)} \right], \tag{24}$$

where $\text{Var}_m[\cdot]$ denotes the sample variance across the M runs. A large V_{phase} indicates that different training runs exhibit substantially different translational motions — precisely the scenario where shift alignment yields the greatest benefit. Without alignment, this run-to-run translational variance would inflate the POD snapshot matrix, requiring additional modes to represent what is essentially a nuisance parameter.

Comparing V_{phase} across experiments reveals which dynamical regimes have the most translational diversity (e.g., high-velocity or variable-speed configurations tend to show larger phase variance).

6.5.5 (e) Autocorrelation of Mean Shift

Panel (e) plots the normalized autocorrelation function (ACF) of the ensemble-mean shift trajectory $\bar{\mathbf{\Delta}}(t)$ defined in (18). For each component c:

$$R_c(\tau) = \frac{\sum_{t=1}^{T-\tau} \big(\bar{\Delta}_c(t) - \mu_c\big)\big(\bar{\Delta}_c(t+\tau) - \mu_c\big)}{\sum_{t=1}^{T} \big(\bar{\Delta}_c(t) - \mu_c\big)^2}, \tag{25}$$

where $\mu_c = \frac{1}{T}\sum_t \bar{\Delta}_c(t)$ is the temporal mean. In the implementation, this is computed via `numpy.correlate` with `mode='full'` and normalized by the zero-lag value $R_c(0) = 1$.

The ACF decays slowly from 1, remaining strongly positive for lags of 30–50 timesteps. This slow decorrelation confirms that the shift sequence is a smooth, persistent process — not white noise. The long correlation length implies that:

- The shift dynamics have strong temporal memory, consistent with a flock maintaining its heading for extended periods.
- A low-order autoregressive model (AR(1)–AR(5)) captures the essential dynamics, as confirmed by panel (c).
- Linear extrapolation of shifts into the forecast period (algorithm 5) is justified for horizons shorter than the decorrelation time.

6.5.6 Summary: Phase–Shape Decomposition

Taken together, panels (a)–(e) establish that the translational shift $\mathbf{\Delta}(t)$ is:

1. **Low-dimensional**: a 2D trajectory in (Δ_x, Δ_y) space.
2. **Smooth**: dominated by low-frequency content (panel b).
3. **Highly predictable**: $R^2 > 0.99$ with AR(3) (panel c).
4. **Persistent**: long autocorrelation time (panel e).

These properties justify the sPOD approach: factor out the simple phase dynamics via registration, and let the POD basis focus exclusively on the remaining *shape dynamics* (deformation, spreading, splitting). The shift prediction at forecast time is a trivial linear extrapolation, while the shape dynamics require the full MVAR/LSTM machinery. This decomposition is the key architectural choice that makes low-dimensional forecasting feasible.

7 Implementation Reference

7.1 Core Functions

Function	Module	Purpose
`fft_cross_correlation_shift`	`shift_align.py`	FFT-based integer pixel shift via (3)
`compute_reference_field`	`shift_align.py`	Reference from mean/first/median
`compute_shifts`	`shift_align.py`	Per-frame shifts for a density stack
`apply_shifts`	`shift_align.py`	Periodic roll (alignment)
`undo_shifts`	`shift_align.py`	Reverse periodic roll (un-alignment)
`align_training_data`	`shift_align.py`	Full training alignment pipeline
`align_test_sequence`	`shift_align.py`	Align test density to training reference
`predict_shifts_linear`	`shift_align.py`	Linear extrapolation of shifts for forecast
`build_pod_basis`	`pod_builder.py`	sPOD construction with optional alignment
`save_pod_basis`	`pod_builder.py`	Serialize basis + shift data to disk
`evaluate_test_runs`	`test_evaluator.py`	Full test-time evaluation loop
`compute_density_grid`	`density.py`	KDE density estimation from particles

7.2 Configuration Keys

The alignment is controlled by YAML configuration:

```yaml
rom:
  shift_align: true       # Enable shift-aligned POD
  shift_align_ref: "mean" # Reference: "mean", "first", "median"
  density_transform: "raw" # "raw", "sqrt", "log", "meansub"
  fixed_modes: 19         # Number of POD modes
  subsample: 1            # Temporal subsampling
```

7.3 Key Code Excerpt: FFT Cross-Correlation

Listing 1: Core FFT cross-correlation (from `shift_align.py`)

```python
def fft_cross_correlation_shift(ref, target):
    Ny, Nx = ref.shape
    F_ref = np.fft.fft2(ref)
    F_target = np.fft.fft2(target)
    cross = np.real(np.fft.ifft2(F_ref * np.conj(F_target)))

    idx = np.unravel_index(np.argmax(cross), cross.shape)
```

```
8
9       dy = idx[0] if idx[0] <= Ny // 2 else idx[0] - Ny
10      dx = idx[1] if idx[1] <= Nx // 2 else idx[1] - Nx
11
12      return int(dy), int(dx)
```

7.4 Key Code Excerpt: Alignment in POD Builder

Listing 2: Shift alignment integration (from `pod_builder.py`)

```
1  if shift_align:
2      Ny, Nx = density_shape_2d
3      densities_2d = X_all.reshape(-1, Ny, Nx)
4      sa_result = align_training_data(
5          densities_2d, M, T_rom, ref_method=shift_align_ref
6      )
7      X_all = sa_result['aligned'].reshape(-1, Ny * Nx)
8      shift_align_data = {
9          'ref': sa_result['ref'],
10         'shifts': sa_result['shifts'],
11         'ref_method': sa_result['ref_method'],
12         'density_shape_2d': density_shape_2d,
13     }
```

8 Discussion

8.1 Strengths

- **Simplicity**: The method requires no learned registration network — only FFT and circular shifts.
- **Exactness**: On a periodic domain, integer shifts incur zero interpolation error.
- **Efficiency**: $O(K \cdot N_y N_x \log N_y N_x)$ for the entire training set.
- **Dramatic improvement**: SVD energy concentration jumps from $\sim$96% to >99.9% at 19 modes, enabling accurate low-dimensional forecasting.

8.2 Limitations and Future Work

- **Integer-only shifts**: Sub-pixel accuracy is not captured. For slowly drifting clusters, this is sufficient; for fast-moving clusters at coarse resolution, sub-pixel registration (e.g., phase-correlation with parabolic interpolation) could help.
- **Translation only**: Rotations, deformations, and scale changes are not handled. A more general approach would use *optimal transport* or *diffeomorphic registration*.
- **Linear shift extrapolation**: The deg = 1 polynomial fit for forecast-period shifts assumes constant velocity. Higher-order extrapolation or a learned shift predictor could improve long-horizon forecasts.
- **Single reference frame**: All snapshots are aligned to one reference, which works well for single-cluster systems. Multi-cluster scenarios would require per-cluster registration or a co-moving frame decomposition.

- **Chicken-and-egg reference**: Computing the reference from unaligned data is suboptimal. An iterative refinement (align → recompute reference → re-align) could converge to a sharper reference field.

9 Conclusion

Shift-aligned POD provides a principled and computationally efficient method for removing translational symmetry from advection-dominated systems on periodic domains. By registering density snapshots to a common reference frame before SVD, the method:

1. Concentrates structural variance into fewer modes (sharper spectral decay).
2. Enables accurate latent forecasting with simple linear (MVAR) or nonlinear (LSTM) models.
3. Provides a clean separation between transport (encoded in shift sequences) and deformation (encoded in latent dynamics).

The XABL ablation confirms that alignment is the single most important preprocessing step, while spectral scaling — seemingly natural normalization — destroys forecast accuracy. These findings guide robust configuration of the Rectsim ROM pipeline for Vicsek collective motion and similar particle-based advection problems.

A Derivation: Cross-Correlation via FFT

The discrete circular cross-correlation of $f, g \in \mathbb{R}^{N_y \times N_x}$ is:

$$(f \star g)[\ell_y, \ell_x] = \sum_{j_y} \sum_{j_x} f[j_y, j_x] \cdot g[(j_y + \ell_y) \bmod N_y,\ (j_x + \ell_x) \bmod N_x].$$

Using the 2D DFT $\hat{f}[\omega_y, \omega_x] = \sum_{j_y, j_x} f[j_y, j_x]\, e^{-2\pi i (j_y \omega_y / N_y + j_x \omega_x / N_x)}$:

$$\begin{aligned}
\widehat{f \star g}[\omega_y, \omega_x] &= \sum_{\ell_y, \ell_x} (f \star g)[\ell_y, \ell_x]\, e^{-2\pi i (\ell_y \omega_y / N_y + \ell_x \omega_x / N_x)} \\
&= \sum_{\ell_y, \ell_x} \sum_{j_y, j_x} f[j_y, j_x]\, g[(j_y + \ell_y),\ (j_x + \ell_x)]\, e^{-2\pi i (\ell_y \omega_y / N_y + \ell_x \omega_x / N_x)} \\
&= \hat{f}[\omega_y, \omega_x] \cdot \overline{\hat{g}[\omega_y, \omega_x]}.
\end{aligned}$$

Therefore: $f \star g = \mathcal{F}^{-1}[\hat{f} \cdot \overline{\hat{g}}]$, which is exactly (3).