CrowdROM: Shift-Aligned POD–MVAR Pipeline

Algorithms for Density-Field Reduced-Order Modeling
with Hellinger Embedding and Simplex Mass Correction

Macias-Olivares, E.

February 2026

1 Problem Setting

We consider N interacting agents on a periodic domain $\Omega = [0, L_x) \times [0, L_y)$ evolving under a discrete-time Vicsek–Morse model (**??**). At each time $t_k = k\,\Delta t$, the particle positions are $\{\mathbf{x}_i^k\}_{i=1}^N \subset \Omega$.

We construct a *density representation* by binning and smoothing:

$$\rho^k(x,y) \;=\; \big(G_\sigma * H^k\big)(x,y), \qquad H_{jl}^k = \frac{1}{\Delta x\,\Delta y}\sum_{i=1}^N \mathbf{1}\big[\mathbf{x}_i^k \in C_{jl}\big], \tag{1}$$

where $H^k \in \mathbb{R}^{N_y \times N_x}$ is the histogram on cells $C_{jl} = [x_j, x_{j+1}) \times [y_l, y_{l+1})$ with spacing $\Delta x = L_x/N_x$, $\Delta y = L_y/N_y$, and G_σ is a 2D Gaussian kernel with bandwidth σ (in grid-cell units), applied with *periodic wrapping* (**?**). The density satisfies $\sum_{jl} \rho_{jl}^k\,\Delta x\,\Delta y = N$.

In code, this is `density.compute_density_grid()` using `scipy.ndimage.gaussian_filter` with `mode='wrap'`.

We flatten each density frame to a column vector $\boldsymbol{\rho}^k \in \mathbb{R}^n$, $n = N_x N_y$, and collect M training trajectories of T subsampled frames each into the *snapshot matrix* $\mathbf{X}_{\text{raw}} = [\boldsymbol{\rho}_1^1, \ldots, \boldsymbol{\rho}_1^T,\ \ldots,\ \boldsymbol{\rho}_M^1, \ldots, \boldsymbol{\rho}_M^T]^\top \in \mathbb{R}^{MT \times n}$.

2 Step 1: Shift Alignment (sPOD)

Particle systems with net translational motion produce density fields that *drift* across Ω. Standard POD is inefficient for traveling structures: a simple translation requires many Fourier-type modes to represent (**??**).

Shifted POD (sPOD) factors out integer-pixel translations *before* computing the POD basis, so that the reduced coordinates only need to capture structural deformation.

2.1 FFT Cross-Correlation Registration

Given a reference field $\rho_{\text{ref}} \in \mathbb{R}^{N_y \times N_x}$ and a target frame ρ^k, we seek the cyclic shift $(\delta_y^k, \delta_x^k) \in \mathbb{Z}^2$ that maximises the cross-correlation:

$$(\delta_y^k, \delta_x^k) \;=\; \underset{(s_y,s_x)\in\mathbb{Z}_{N_y}\times\mathbb{Z}_{N_x}}{\arg\max} \sum_{j,l} \rho_{\text{ref}}(j,l)\,\rho^k\big((j+s_y) \mod N_y,\ (l+s_x) \mod N_x\big). \tag{2}$$

Algorithm 1 Shift Alignment (sPOD) — Training

Require: Snapshot matrix $\mathbf{X}_{\text{raw}} \in \mathbb{R}^{MT\times n}$ reshaped as $\{\rho^k\}_{k=1}^{MT}$, each $\rho^k \in \mathbb{R}^{N_y\times N_x}$
Ensure: Aligned snapshots $\{\tilde{\rho}^k\}$, reference ρ_{ref}, shift array $\boldsymbol{\Delta} \in \mathbb{Z}^{MT\times 2}$

```
1: $\rho_{\text{ref}} \leftarrow \frac{1}{MT}\sum_{k=1}^{MT}\rho^k$                                  ▷ shift_align_ref='mean'
2: for $k = 1,\dots,MT$ do
3:     $\hat{F}_{\text{ref}} \leftarrow \texttt{fft2}(\rho_{\text{ref}})$;  $\hat{F}_k \leftarrow \texttt{fft2}(\rho^k)$
4:     $C \leftarrow \mathrm{Re}\big(\texttt{ifft2}(\hat{F}_{\text{ref}} \odot \bar{\hat{F}}_k)\big)$
5:     $(\delta_y^k, \delta_x^k) \leftarrow \arg\max_{(s_y,s_x)} C_{s_y,s_x}$                            ▷ wrap to signed
6:     $\tilde{\rho}^k \leftarrow \texttt{roll}(\rho^k,\ \delta_y^k,\ \delta_x^k)$                              ▷ Eq. (??)
7: end for
8: return $\{\tilde{\rho}^k\}$, $\rho_{\text{ref}}$, $\boldsymbol{\Delta}$
```

Algorithm 2 Shift Alignment — Test-Time Un-alignment

Require: Test density $\rho_{\text{test}}^{1:T_{\text{test}}}$, training reference ρ_{ref}, forecast start index T_0
Ensure: Aligned test density, predicted shifts for forecast

```
1: for $k = 1,\dots,T_{\text{test}}$ do
2:     $(\delta_y^k, \delta_x^k) \leftarrow \texttt{fft\_cross\_correlation\_shift}(\rho_{\text{ref}}, \rho_{\text{test}}^k)$
3:     $\tilde{\rho}_{\text{test}}^k \leftarrow \texttt{roll}(\rho_{\text{test}}^k, \delta_y^k, \delta_x^k)$
4: end for
5: Fit $\hat{\boldsymbol{\delta}}(t) = \mathbf{c}_0 + \mathbf{c}_1 t$ to $\{\boldsymbol{\delta}^k\}_{k=1}^{T_0}$ via least-squares          ▷ predict_shifts_linear
6: Extrapolate $\hat{\boldsymbol{\delta}}^k$ for $k = T_0+1,\dots,T_0+H$
7: Un-align predictions: $\hat{\rho}_{\text{phys}}^k \leftarrow \texttt{roll}(\hat{\tilde{\rho}}^k, -\hat{\delta}_y^k, -\hat{\delta}_x^k)$
```

By the convolution theorem on the periodic domain, this is computed in $O(n\log n)$ via the 2D FFT:

$$C = \mathcal{F}^{-1}\big[\mathcal{F}[\rho_{\text{ref}}] \odot \overline{\mathcal{F}[\rho^k]}\big], \qquad (\delta_y^k, \delta_x^k) = \underset{(s_y,s_x)}{\arg\max}\,\mathrm{Re}(C_{s_y,s_x}), \tag{3}$$

where $\overline{(\cdot)}$ denotes elementwise complex conjugation and $\odot$ is the Hadamard product. The shift is converted to signed integers by wrapping indices exceeding $N_y/2$ or $N_x/2$.

2.2 Alignment and Un-alignment

The aligned frame is obtained by cyclic rolling:

$$\tilde{\rho}^k(j,l) = \rho^k\big((j-\delta_y^k) \mod N_y,\ (l-\delta_x^k) \mod N_x\big). \tag{4}$$

At prediction time, forecasted (aligned) fields are *un-aligned* by applying the *negative* shift:

$$\hat{\rho}_{\text{phys}}^k(j,l) = \hat{\tilde{\rho}}^k\big((j+\delta_y^k) \mod N_y,\ (l+\delta_x^k) \mod N_x\big). \tag{5}$$

For forecast frames beyond the teacher-forced window, shifts are *linearly extrapolated*: $\hat{\boldsymbol{\delta}}(t) = \mathbf{c}_0 + \mathbf{c}_1 t$, fit by least-squares to the known shifts.

Implementation. Defined in `src/rectsim/shift_align.py`: `fft_cross_correlation_shift()`, `compute_reference_field()`, `align_training_data()`, `apply_shifts()`/`undo_shifts()`, `predict_shifts_linea`
Called from `pod_builder.build_pod_basis()` (training) and `test_evaluator.evaluate_test_runs()` (testing). Config keys: `rom.shift_align: true`, `rom.shift_align_ref: "mean"`.

3 Step 2: Hellinger (Square-Root) Density Transform

After alignment, we apply a variance-stabilising transform before POD. The *square-root transform* embeds densities into the *Hellinger space* (**??**):

$$f(\boldsymbol{\rho}) = \sqrt{\boldsymbol{\rho} + \varepsilon}, \qquad \varepsilon = 10^{-10}, \tag{6}$$

applied element-wise. The small ε prevents division issues at zero-density cells.

3.1 Why It Helps POD

1. **Dynamic range compression.** Raw density fields are non-negative and spiky: a few bright cluster peaks dominate the L^2 variance. The square root compresses peaks ($\sqrt{\rho_{\max}}$) and lifts tails ($\sqrt{\rho_{\min}}$), distributing energy across more POD modes and requiring fewer modes for a given energy threshold.

2. **Hellinger metric.** In the transformed space, the L^2 distance equals the Hellinger distance (**?**):
$$\left\|\sqrt{\rho_1} - \sqrt{\rho_2}\right\|_2^2 = H^2(\rho_1, \rho_2) = \frac{1}{2}\int \left(\sqrt{\rho_1} - \sqrt{\rho_2}\right)^2, \tag{7}$$
which is a proper statistical divergence between density functions, bounded in $[0, 1]$ for probability measures. Minimising POD reconstruction error in Hellinger space is therefore statistically principled.

3. **Variance stabilisation.** For Poisson-like count data (as in histogrammed particles), $\mathrm{Var}[\rho] \propto \rho$. The Anscombe transform $\sqrt{\rho + 3/8}$ is the classical variance-stabiliser (**?**). Our $\sqrt{\rho + \varepsilon}$ serves the same role, making the POD residual approximately homoscedastic.

3.2 Inverse Transform

After POD lifting, the reconstructed field $\hat{\mathbf{y}}$ lies in the *transformed* space. The inverse is:

$$\hat{\boldsymbol{\rho}} = \left[\max(\hat{\mathbf{y}},\ 0)\right]^2 - \varepsilon. \tag{8}$$

The $\max(\cdot, 0)$ clamp is necessary because POD reconstruction (a linear projection) can produce negative values in the transformed space, even though true $\sqrt{\rho + \varepsilon} > 0$.

Implementation. **Forward:** `pod_builder.py` lines 118–120: `X_all = np.sqrt(X_all + density_transform_`
Applied *after* shift alignment, *before* POD mean-centering. **Inverse:** `test_evaluator.py` lines 355–357: `np.maximum(pred, 0.0)**2 - density_transform_eps`. Config key: `rom.density_transform: "sqrt"`, `rom.density_transform_eps: 1e-10`.

4 Step 3: Proper Orthogonal Decomposition (POD)

Given the aligned, transformed snapshot matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{MT \times n}$:

1. **Mean-center:** $\boldsymbol{\mu} = \frac{1}{MT} \sum_k \tilde{\mathbf{x}}_k, \quad \bar{\mathbf{X}} = \tilde{\mathbf{X}} - \mathbf{1}\boldsymbol{\mu}^\top$.
2. **SVD:** $\bar{\mathbf{X}}^\top = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top, \quad \mathbf{U} \in \mathbb{R}^{n \times MT}, \ \sigma_1 \geq \sigma_2 \geq \cdots \geq 0$.
3. **Truncate** to d modes, determined by either:
   - *Fixed:* $d =$ `rom.fixed_modes` (e.g. $d = 19$), or
   - *Energy threshold:* smallest d such that $\sum_{i=1}^{d} \sigma_i^2 \big/ \sum_{i=1}^{\min(MT,n)} \sigma_i^2 \geq E_{\text{target}}$ (e.g. $E_{\text{target}} = 0.99$).

   $\mathbf{U}_d = [\mathbf{u}_1, \ldots, \mathbf{u}_d] \in \mathbb{R}^{n \times d}$.
4. **Project to latent space:** $\mathbf{a}^k = \mathbf{U}_d^\top (\tilde{\mathbf{x}}_k - \boldsymbol{\mu}) \in \mathbb{R}^d$.
5. **Lift (reconstruct):** $\hat{\tilde{\mathbf{x}}}_k = \mathbf{U}_d \, \hat{\mathbf{a}}^k + \boldsymbol{\mu}$.

Implementation. `pod_builder.build_pod_basis()`: uses `np.linalg.svd(X_centered.T, full_matrices=Fa` Fixed modes from `rom.fixed_modes`, energy from `rom.pod_energy`.

5 Step 4: MVAR Latent Dynamics

In the latent space $\{\mathbf{a}^k \in \mathbb{R}^d\}$, we model the dynamics with a *Vector AutoRegressive model of order* p (VAR(p)) with ridge regularisation (**??**):

$$\mathbf{a}^{k+1} = \sum_{j=1}^{p} \mathbf{A}_j \, \mathbf{a}^{k+1-j} + \mathbf{b} + \boldsymbol{\epsilon}^k, \qquad \boldsymbol{\epsilon}^k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\epsilon), \tag{9}$$

where $\mathbf{A}_j \in \mathbb{R}^{d \times d}$ are the coefficient matrices, $\mathbf{b} \in \mathbb{R}^d$ is the intercept, and p is the lag order.

5.1 Training

Stack the lagged history into a design row:

$$\mathbf{x}_{\text{train}}^k = \left[\mathbf{a}^{k-p}, \ \mathbf{a}^{k-p+1}, \ \ldots, \ \mathbf{a}^{k-1}\right]^\top \in \mathbb{R}^{pd}, \qquad \mathbf{y}_{\text{train}}^k = \mathbf{a}^k \in \mathbb{R}^d. \tag{10}$$

The coefficient matrix $\mathbf{W} = [\mathbf{A}_p, \ldots, \mathbf{A}_1, \mathbf{b}] \in \mathbb{R}^{d \times (pd+1)}$ is found by ridge regression:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_k \left\| \mathbf{y}^k - \mathbf{W} \, [\mathbf{x}^k; \, 1] \right\|_2^2 + \alpha \|\mathbf{W}\|_F^2, \tag{11}$$

with regularisation $\alpha > 0$ (config: `rom.models.mvar.ridge_alpha`). This has the closed-form solution via ridge normal equations, implemented using `sklearn.linear_model.Ridge(fit_intercept=True)`.

Algorithm 3 Euclidean Projection onto the Scaled Simplex $\mathcal{S}_{M_0}$

Require: $\hat{\boldsymbol{\rho}} \in \mathbb{R}^n$, mass target $M_0 > 0$
Ensure: $\boldsymbol{\rho}^* = \Pi_{\mathcal{S}_{M_0}}(\hat{\boldsymbol{\rho}})$

```
1: $\mathbf{v} \leftarrow \hat{\boldsymbol{\rho}}/M_0$                                      ▷ scale to unit simplex
2: Sort: $u_1 \geq u_2 \geq \cdots \geq u_n$ (descending)
3: $S_j \leftarrow \sum_{i=1}^{j} u_i - 1, \quad j = 1, \ldots, n$                ▷ running cumulative sum
4: $\hat{j} \leftarrow \max\{j \in [n] \mid u_j - S_j/j > 0\}$
5: $\theta \leftarrow S_{\hat{j}}/\hat{j}$                                       ▷ dual variable / Lagrange multiplier
6: $\rho_i^* \leftarrow \max(v_i - \theta,\ 0) \cdot M_0, \quad i = 1, \ldots, n$
7: return $\boldsymbol{\rho}^*$
```

5.2 Autoregressive Forecasting

Given the last p known latent states $\mathbf{a}^{T_0-p+1}, \ldots, \mathbf{a}^{T_0}$ (the "initial condition window"), the forecast is generated autoregressively:

$$\hat{\mathbf{a}}^{T_0+h} = \sum_{j=1}^{p} \mathbf{A}_j \, \hat{\mathbf{a}}^{T_0+h-j} + \mathbf{b}, \qquad h = 1, 2, \ldots, H, \tag{12}$$

where $\hat{\mathbf{a}}^k = \mathbf{a}^k$ for $k \leq T_0$ (teacher-forced prefix).

Implementation. `src/rectsim/mvar_trainer.py`: `train_mvar_model()`. Config: `rom.models.mvar.`{`lag`, `ridge_alpha`}.

6 Step 5: Simplex Mass Projection

The nonlinear inverse transform ($\hat{\rho} = \max(\hat{y}, 0)^2 - \varepsilon$) does *not* preserve total mass: $\sum_i \hat{\rho}_i \neq \sum_i \rho_i^0 = M_0$ because the square of a linear reconstruction is not linear (Jensen's inequality). We correct this by an L^2-optimal projection onto the *scaled probability simplex* (**?**):

$$\mathcal{S}_{M_0} = \{\boldsymbol{\rho} \in \mathbb{R}^n \mid \rho_i \geq 0 \, \forall \, i, \quad \textstyle\sum_{i=1}^n \rho_i = M_0\}, \tag{13}$$

where $M_0 = \sum_i \rho_i^{T_0}$ is the ground-truth total mass at the forecast start.

$$\Pi_{\mathcal{S}_{M_0}}(\hat{\boldsymbol{\rho}}) = \underset{\boldsymbol{\rho} \in \mathcal{S}_{M_0}}{\arg\min} \|\boldsymbol{\rho} - \hat{\boldsymbol{\rho}}\|_2^2. \tag{14}$$

6.1 Algorithm (Duchi et al., 2008)

Complexity. $O(n \log n)$ due to the sort; all other operations are $O(n)$.

KKT interpretation. The Lagrangian of (**??**) on the unit simplex is $L(\boldsymbol{\rho}, \lambda, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{\rho} - \hat{\boldsymbol{\rho}}\|^2 + \lambda(\mathbf{1}^\top \boldsymbol{\rho} - M_0) - \boldsymbol{\mu}^\top \boldsymbol{\rho}$. The KKT conditions yield $\rho_i^* = \max(\hat{\rho}_i - \lambda^*, 0)$ with $\lambda^* = \theta \cdot M_0$ chosen so that $\sum_i \rho_i^* = M_0$. When $\hat{\boldsymbol{\rho}} \geq 0$ already (as after the $\max(\cdot, 0)^2$ step), the projection reduces to a *uniform additive shift* $\rho_i^* = \hat{\rho}_i - \lambda^*$ with truncation at zero, identical to the offset-then-clamp method.

Implementation. `test_evaluator._project_simplex(rho_frame, mass_target)`. Applied per frame after the inverse $\sqrt{\rho}$ transform. Config: `rom.mass_postprocess: "simplex"`.

7 Complete Pipeline

7.1 Training Pipeline

1. **Simulate** M training trajectories (Vicsek–Morse, N agents, T_{sim} seconds). `cli.py`
2. **KDE**: bin particles onto $N_y \times N_x$ grid, Gaussian-smooth with bandwidth σ. `density.py`

$$\boldsymbol{\rho}^k \in \mathbb{R}^{N_y \times N_x}, \qquad \sum_{j,l} \rho_{jl}\, \Delta x\, \Delta y = N. \tag{15}$$

3. **Shift Alignment** (Alg. **??**): compute reference ρ_{ref} and per-frame shifts $\boldsymbol{\delta}^k$; roll all frames to remove translational drift. `shift_align.py`

$$\tilde{\boldsymbol{\rho}}^k = \mathcal{T}_{\boldsymbol{\delta}^k}[\boldsymbol{\rho}^k]. \tag{16}$$

4. $\sqrt{\rho}$ **Transform**: embed aligned densities into Hellinger space. `pod_builder.py`

$$\tilde{\mathbf{x}}^k = \sqrt{\tilde{\boldsymbol{\rho}}^k + \varepsilon}. \tag{17}$$

5. **POD (SVD)**: mean-centre, compute SVD, truncate to d modes. `pod_builder.py`

$$\boldsymbol{\mu} = \overline{\tilde{\mathbf{x}}}, \qquad (\tilde{\mathbf{X}} - \mathbf{1}\boldsymbol{\mu}^\top)^\top = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top, \qquad \mathbf{a}^k = \mathbf{U}_d^\top(\tilde{\mathbf{x}}^k - \boldsymbol{\mu}). \tag{18}$$

6. **MVAR**: fit VAR(p) with ridge α on latent trajectories $\{\mathbf{a}^k\}$. `mvar_trainer.py`

$$\mathbf{W}^* = \arg\min \sum_k \|\mathbf{a}^{k+1} - \mathbf{W}[\mathbf{a}^{k-p+1:k};\, 1]\|^2 + \alpha\|\mathbf{W}\|_F^2. \tag{19}$$

7.2 Prediction Pipeline (Test Time)

Given a test trajectory with density known up to $t = t_0$ (the "conditioning window"):

1. **Shift-align** test density to training reference: $\tilde{\rho}^k_{\text{test}}$. Record shifts $\boldsymbol{\delta}^{1:T_0}_{\text{test}}$. `test_evaluator.py`
2. $\sqrt{\rho}$ **forward**: $\tilde{x}^k_{\text{test}} = \sqrt{\tilde{\rho}^k_{\text{test}} + \varepsilon}$.
3. **Project to latent**: $\mathbf{a}^k_{\text{test}} = \mathbf{U}_d^\top(\tilde{x}^k_{\text{test}} - \boldsymbol{\mu})$.
4. **MVAR forecast**: autoregressively generate $\hat{\mathbf{a}}^{T_0+1}, \ldots, \hat{\mathbf{a}}^{T_0+H}$.
5. **Lift (POD inverse)**: $\hat{\mathbf{y}}^k = \mathbf{U}_d\, \hat{\mathbf{a}}^k + \boldsymbol{\mu}$.
6. **Inverse** $\sqrt{\rho}$ (Eq. **??**): $\hat{\tilde{\boldsymbol{\rho}}}^k = [\max(\hat{\mathbf{y}}^k, 0)]^2 - \varepsilon$.
7. **Simplex projection** (Alg. **??**): $\hat{\tilde{\boldsymbol{\rho}}}^k \leftarrow \Pi_{\mathcal{S}_{M_0}}(\hat{\tilde{\boldsymbol{\rho}}}^k)$.
8. **Un-align shift**: extrapolate shifts for forecast horizon, reverse the cyclic roll. $\hat{\boldsymbol{\rho}}^k_{\text{phys}} = \mathcal{T}_{-\hat{\boldsymbol{\delta}}^k}[\hat{\tilde{\boldsymbol{\rho}}}^k]$.

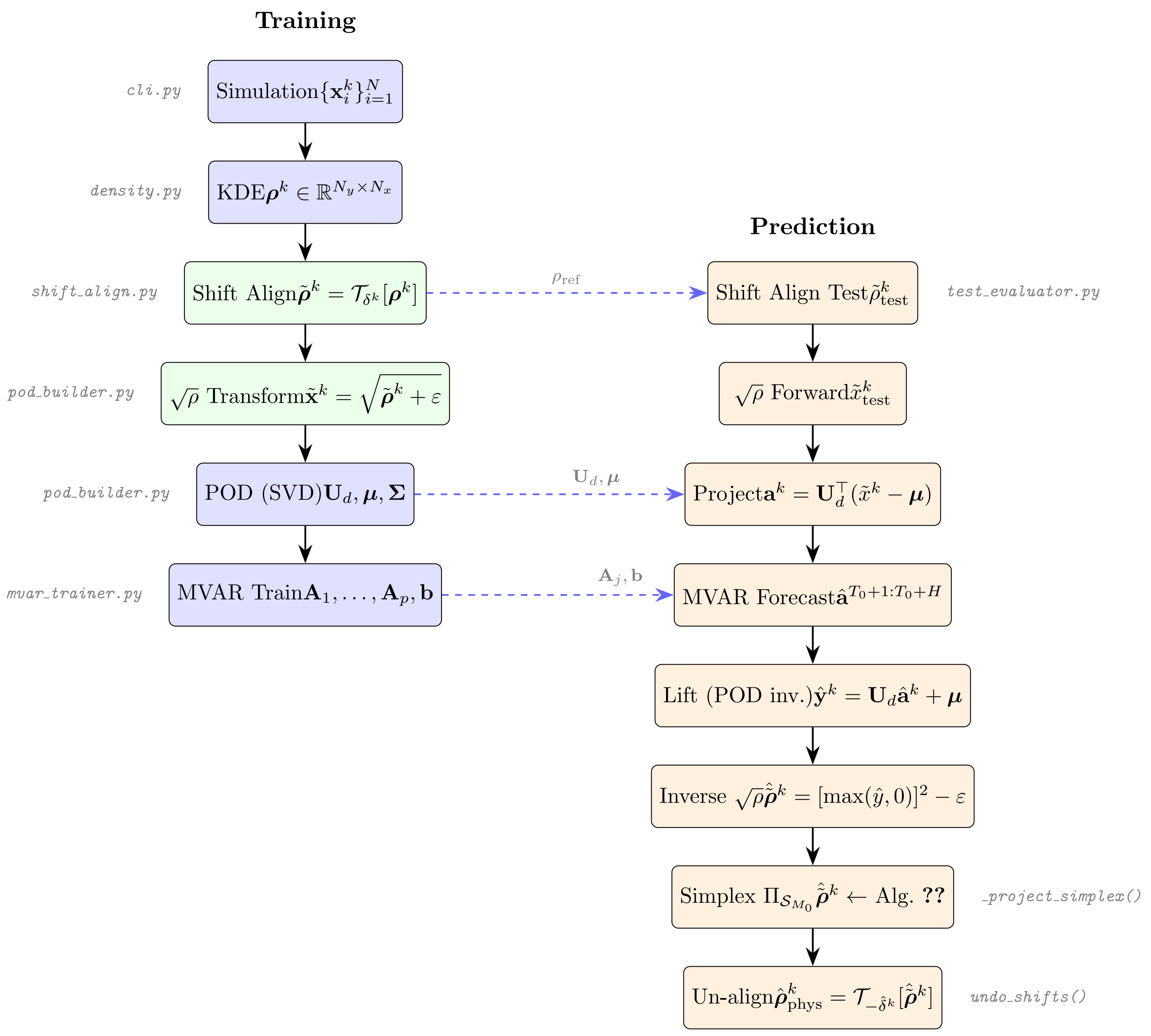
Training
cli.py
Simulation$\{\mathbf{x}_i^k\}_{i=1}^N$
density.py
KDE$\boldsymbol{\rho}^k \in \mathbb{R}^{N_y \times N_x}$
shift_align.py
Shift Align$\tilde{\boldsymbol{\rho}}^k = \mathcal{T}_{\delta^k}[\boldsymbol{\rho}^k]$
pod_builder.py
$\sqrt{\rho}$ Transform$\tilde{\mathbf{x}}^k = \sqrt{\tilde{\boldsymbol{\rho}}^k + \varepsilon}$
pod_builder.py
POD (SVD)$\mathbf{U}_d, \boldsymbol{\mu}, \boldsymbol{\Sigma}$
mvar_trainer.py
MVAR Train$\mathbf{A}_1, \ldots, \mathbf{A}_p, \mathbf{b}$
ρ_{ref}
$\mathbf{U}_d, \boldsymbol{\mu}$
$\mathbf{A}_j, \mathbf{b}$
Prediction
Shift Align Test$\tilde{\rho}^k_{\text{test}}$
test_evaluator.py
$\sqrt{\rho}$ Forward$\tilde{x}^k_{\text{test}}$
Project$\mathbf{a}^k = \mathbf{U}_d^\top(\tilde{x}^k - \boldsymbol{\mu})$
MVAR Forecast$\hat{\mathbf{a}}^{T_0+1:T_0+H}$
Lift (POD inv.)$\hat{\mathbf{y}}^k = \mathbf{U}_d\hat{\mathbf{a}}^k + \boldsymbol{\mu}$
Inverse $\sqrt{\rho}\hat{\tilde{\boldsymbol{\rho}}}^k = [\max(\hat{y}, 0)]^2 - \varepsilon$
Simplex $\Pi_{\mathcal{S}_{M_0}}\hat{\tilde{\boldsymbol{\rho}}}^k \leftarrow$ Alg. ??
_project_simplex()
Un-align$\hat{\boldsymbol{\rho}}^k_{\text{phys}} = \mathcal{T}_{-\hat{\delta}^k}[\hat{\tilde{\boldsymbol{\rho}}}^k]$
undo_shifts()

Figure 1: Complete CrowdROM pipeline. Blue = training only, orange = test time only, green = shared operations. Dashed arrows show training artifacts consumed at test time.

7.3 Pipeline Diagram

8 Configuration Summary

All pipeline options are set in the YAML config under the `rom:` block:

Config Key	Values	Effect
`shift_align`	`true/false`	Enable sPOD alignment
`shift_align_ref`	`"mean"`, `"first"`, `"median"`	Reference field computation
`density_transform`	`"raw"`, `"sqrt"`, `"log"`	Pre-POD transform
`density_transform_eps`	float (10^{-10})	Regularisation ε
`fixed_modes`	int or `null`	Fixed POD rank d
`pod_energy`	float (0.99)	Energy threshold if no fixed rank
`mass_postprocess`	`"none"`, `"simplex"`, `"scale"`	Post-inverse mass correction
`models.mvar.lag`	int (5)	VAR order p
`models.mvar.ridge_alpha`	float (10^{-4})	Ridge regularisation α

Table 1: Pipeline configuration keys (`rom:` section of YAML).

References

Anscombe, F. J. (1948). The transformation of Poisson, binomial and negative-binomial data. *Biometrika*, 35(3/4):246–254.

Black, F., Schulze, P., and Reiss, J. (2020). Projection-based model reduction with dynamically transformed modes. *ESAIM: Mathematical Modelling and Numerical Analysis*, 54(6):2011–2043.

D'Orsogna, M. R., Chuang, Y.-L., Bertozzi, A. L., and Chayes, L. S. (2006). Self-propelled particles with soft-core interactions: patterns, stability, and collapse. *Physical Review Letters*, 96(10):104302.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 272–279.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer, 2nd edition.

Le Cam, L. (1986). *Asymptotic Methods in Statistical Decision Theory*. Springer-Verlag, New York.

Lütkepohl, H. (2005). *New Introduction to Multiple Time Series Analysis*. Springer-Verlag, Berlin.

Reiss, J., Schulze, P., Sesterhenn, J., and Mehrmann, V. (2018). The shifted proper orthogonal decomposition: A mode decomposition for multiple transport phenomena. *SIAM Journal on Scientific Computing*, 40(3):A1322–A1344.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London.

Tsybakov, A. B. (2009). *Introduction to Nonparametric Estimation*. Springer, New York.

Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., and Shochet, O. (1995). Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6):1226–1229.