# State Estimation of the Multi-Scale Lorenz 96 System
# A Hybrid ESN + EnKF Implementation

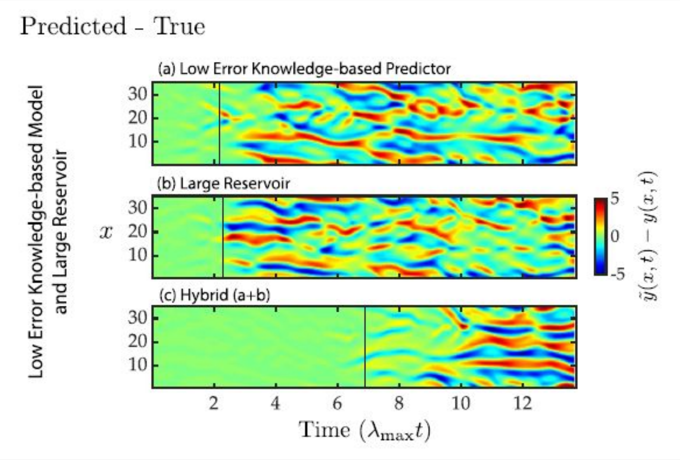
Group 1

Marcus, Maxwell, Anderson, Josiah, Mohamed

# Motivation

**Inspiration**: Papers on Hybrid ESN Implementation, EnKF Implementation

**Hypothesis:** We can combine the strengths of Hybrid ESN methodologies and Ensemble Kalman Filtering to create an even stronger model



# Basic Overview

**Multi-scale Lorenz 96:** Couples “slow” large-scale variables with “fast” small-scale variables

**Echo State Network:** An easy-to-implement type of LSM that uses randomly generated input weights and reservoir layer to learn nonlinear dynamics

**Ensemble Kalman Filter:** Data assimilation method that updates an ensemble of model forecasts by blending observations with model predictions using Kalman-filter principles

# Putting it all together

**System:** Multi-scale Lorenz 96

**Model Architecture:** (Hybrid) Echo State Network

**Data Assimilation Method:** Ensemble Kalman Filter

**Data:** Multi-Scale Synthetic Data Generated from True Equations

**Training:** True Data and Imperfect Model

# Tested Models

| | |
|---|---|
| Imperfect Model ✅ | Imperfect Model w/ EnKF ✅ |
| ESN ✅ | ESN w/ EnKF ✅ |
| Hybrid ESN ❌ | Hybrid ESN w/ EnKF ❌ |

# Data Generation

$$\frac{dX_k}{dt} = X_{k-1}(X_{k+1} - X_{k-2}) - X_k + F - \frac{hc}{b}\sum_{j=1}^{J} Y_{j,k}$$

$$\frac{dY_{j,k}}{dt} = -cb \cdot Y_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} + \frac{hc}{b}X_k - \frac{he}{d}\sum_{i=1}^{I} Z_{i,j,k}$$

$$\frac{dZ_{i,j,k}}{dt} = ed \cdot Z_{i-1,j,k}(Z_{i+1,j,k} - Z_{i-2,j,k}) - eZ_{i,j,k} + \frac{he}{d}Y_{j,k}$$

# Data Generation

$$X^{(t+\Delta t)} = \mathrm{RK4}\left(\frac{dX}{dt}, \Delta t\right), \quad X \in \mathbb{R}^{8\times 1}$$

$$Y^{(t+\Delta t)} = \mathrm{RK4}\left(\frac{dY}{dt}, \Delta t\right), \quad Y \in \mathbb{R}^{8\times 8}$$

$$Z^{(t+\Delta t)} = \mathrm{RK4}\left(\frac{dZ}{dt}, \Delta t\right), \quad Z \in \mathbb{R}^{8\times 8\times 8}$$

# Data Generation

$$X_k(0) \sim \text{UniformInteger}([-5, 4]) \quad \text{for } k = 1, \ldots, 8$$

$$Y_{j,k}(0) \sim \mathcal{N}(0, 1) \quad \text{for } j, k = 1, \ldots, 8$$

$$Z_{i,j,k}(0) \sim \mathcal{N}(0, 0.05^2) \quad \text{for } i, j, k = 1, \ldots, 8$$

# Data Generation

$$X_{\text{norm}} = \frac{X - \mu_X}{\sigma_X}, \quad \mu_X = \frac{1}{T}\sum_{t=1}^{T} X_t, \quad \sigma_X = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(X_t - \mu_X)^2}$$

$$Y_{\text{norm}} = \frac{Y - \mu_Y}{\sigma_Y}, \quad \mu_Y = \frac{1}{T}\sum_{t=1}^{T} Y_t, \quad \sigma_Y = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(Y_t - \mu_Y)^2}$$

$$\text{data}_{\text{norm}} = [X_{\text{norm}} \parallel Y_{\text{norm}}] \in \mathbb{R}^{T \times (K + JK)}$$

# Imperfect Model

$$\frac{dX_k}{dt} = X_{k-1}(X_{k+1} - X_{k-2}) - X_k + F - \frac{hc}{b}\sum_{j=1}^{J} Y_{j,k}$$

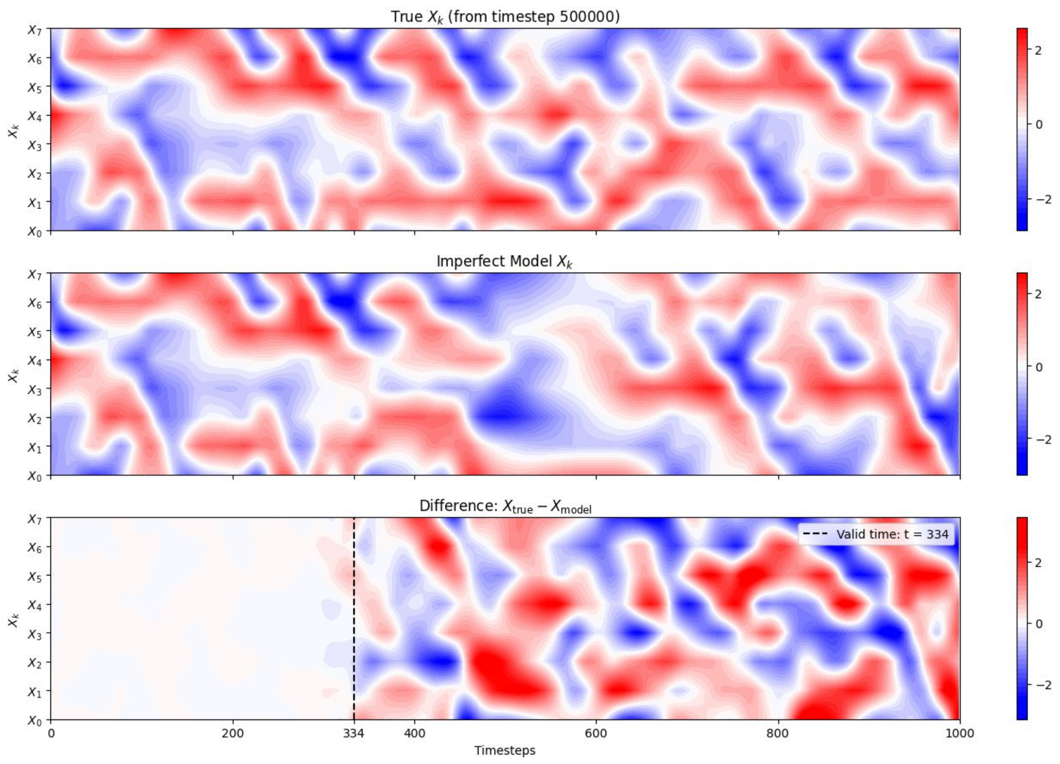
$$\frac{dY_{j,k}}{dt} = -cb \cdot Y_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} + \frac{hc}{b}X_k$$

$$X = X_{\text{norm}} \cdot \sigma_X + \mu_X$$
$$Y = Y_{\text{norm}} \cdot \sigma_Y + \mu_Y$$

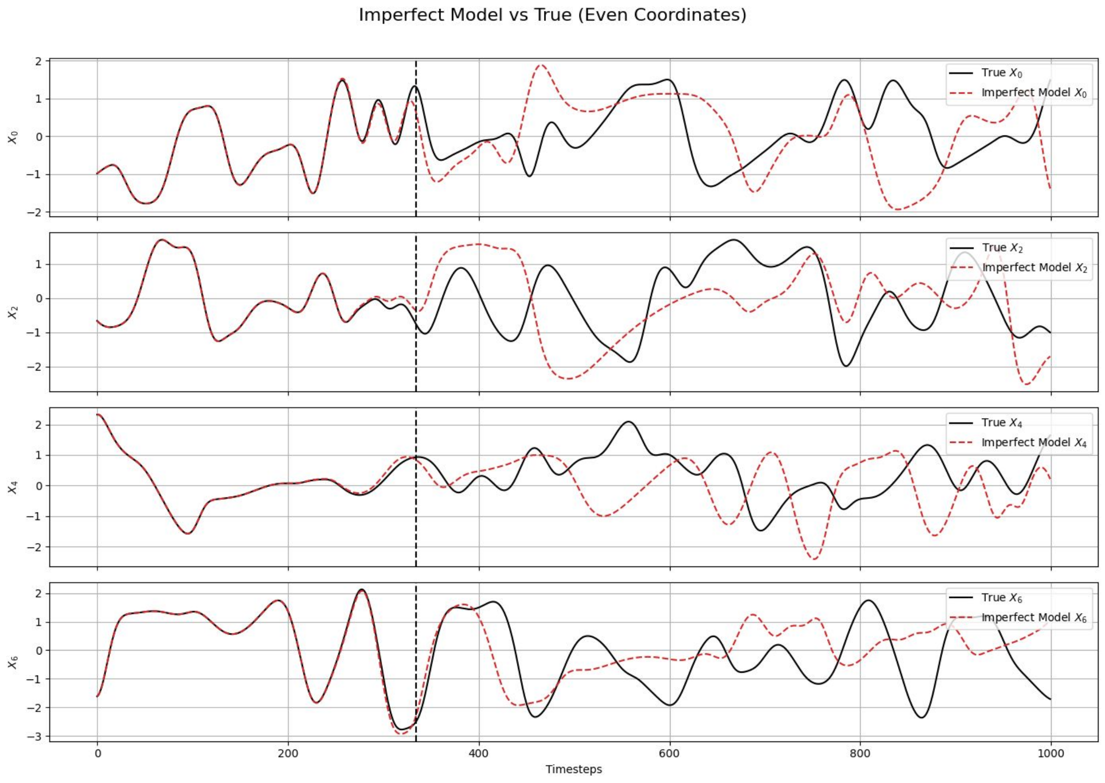
$$(X', Y') = \text{RK4}\left(\frac{dX}{dt}, \frac{dY}{dt}, \Delta t\right)$$

$$X'_{\text{norm}} = \frac{X' - \mu_X}{\sigma_X}$$
$$Y'_{\text{norm}} = \frac{Y' - \mu_Y}{\sigma_Y}$$

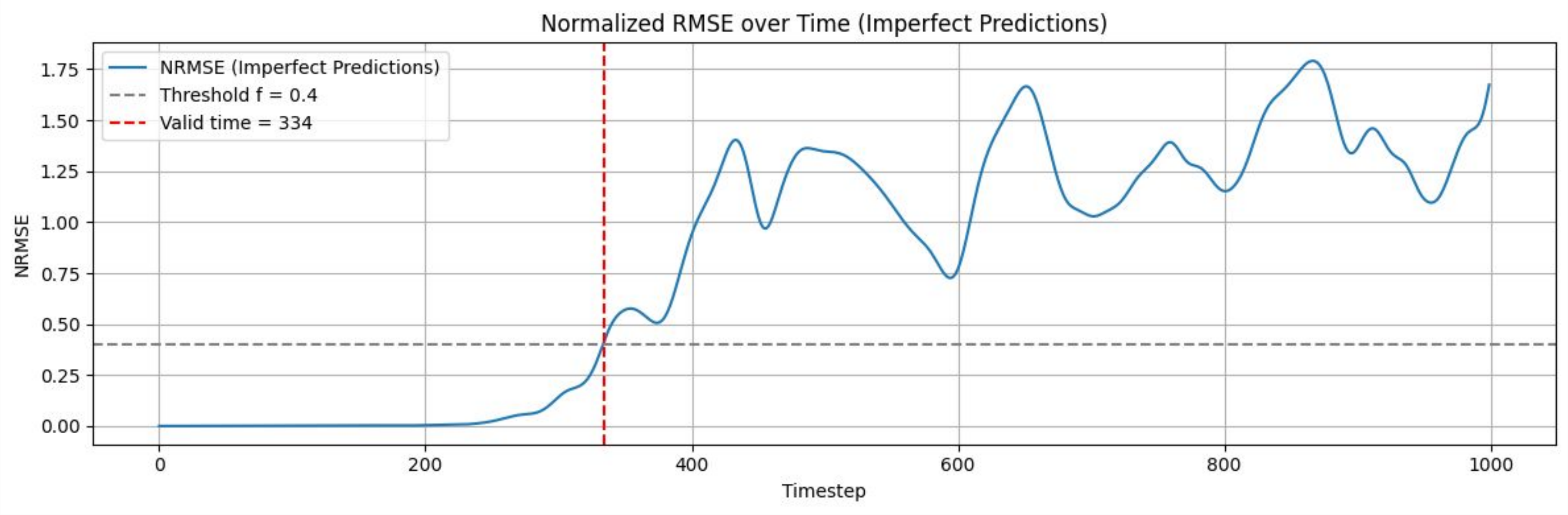
# Imperfect Model



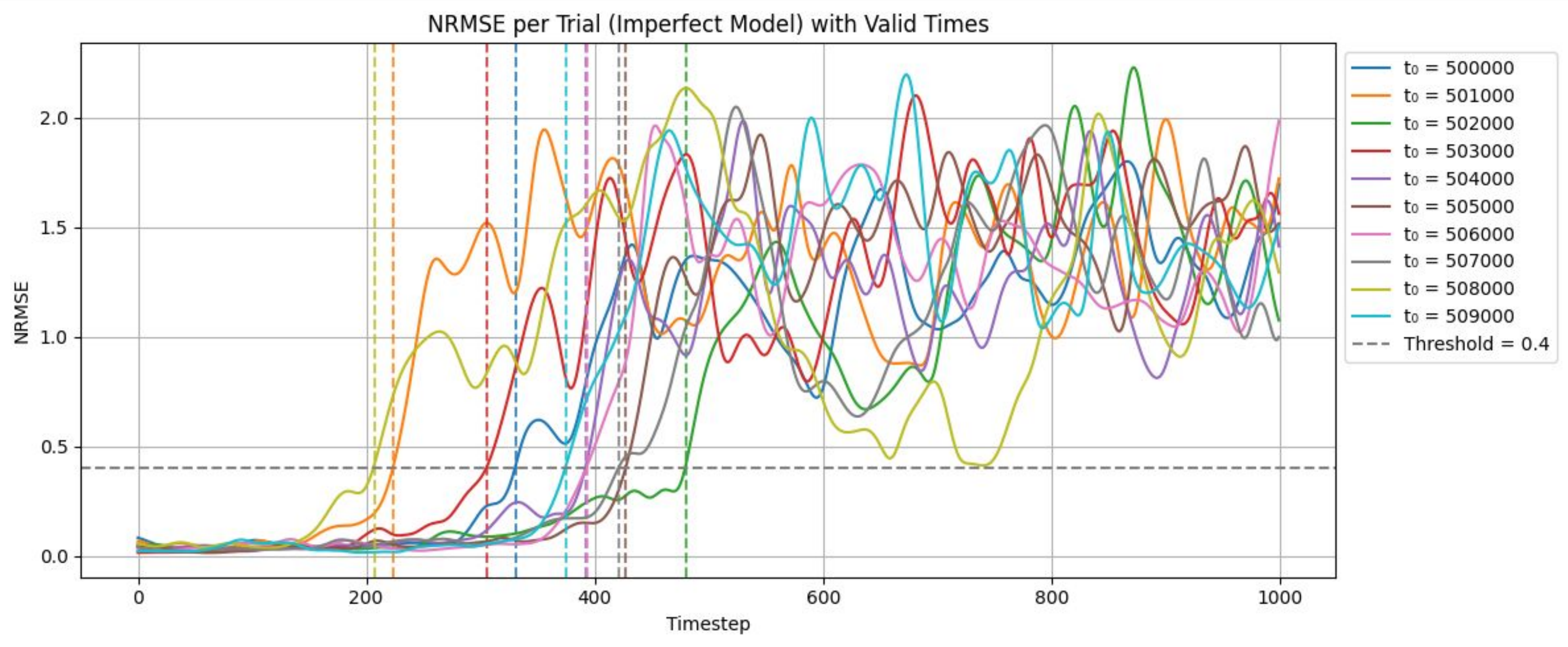
# Imperfect Model



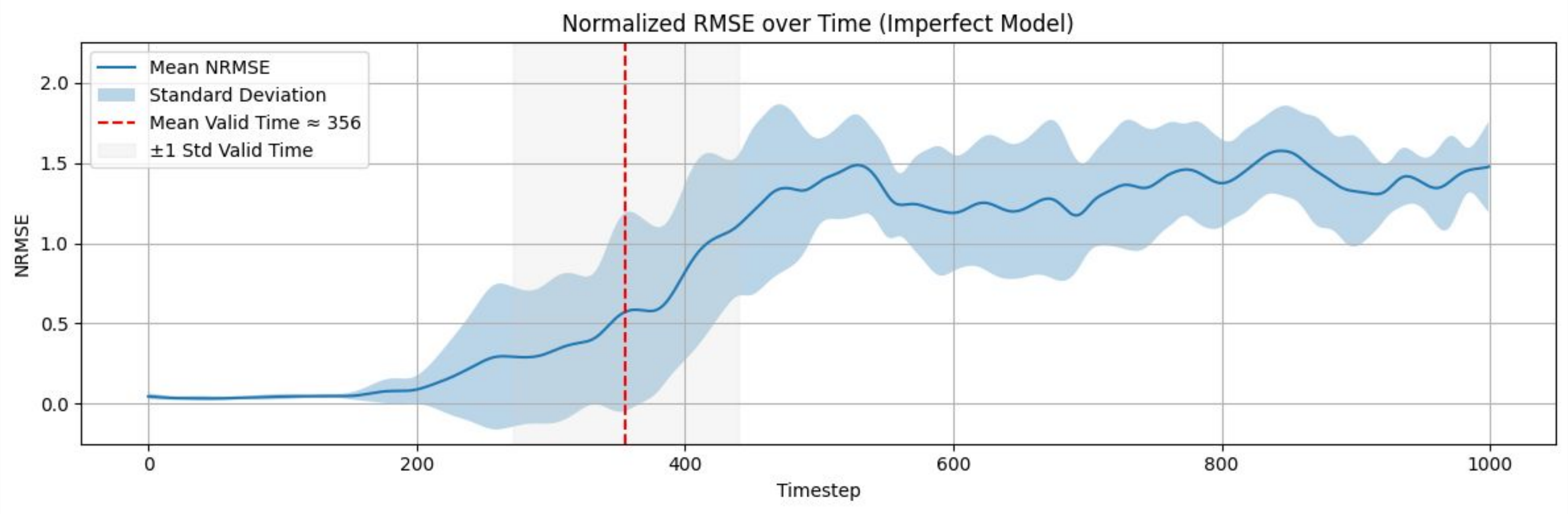
# Imperfect Model



# Imperfect Model



# Imperfect Model



# Echo State Network

Inspired by neuroscientific models
Difficulty of training RNNs
Lack of convergence in E-BP

$$\tilde{x}(n) = \tanh(\mathbf{W}^{in}[1; \mathbf{u}(n)] + \mathbf{W}x(n-1))$$
$$x(n) = (1-\alpha)x(n-1) + \alpha\tilde{x}(n)$$

$$\mathbf{W}^{out} = \mathbf{Y}^{target}\mathbf{X}^T\left(\mathbf{X}\mathbf{X} + \beta\mathbf{I}\right)^{-1}$$

$$y(n) = \mathbf{W}^{out}[1; \mathbf{u}(n); \mathbf{x}(n)]$$

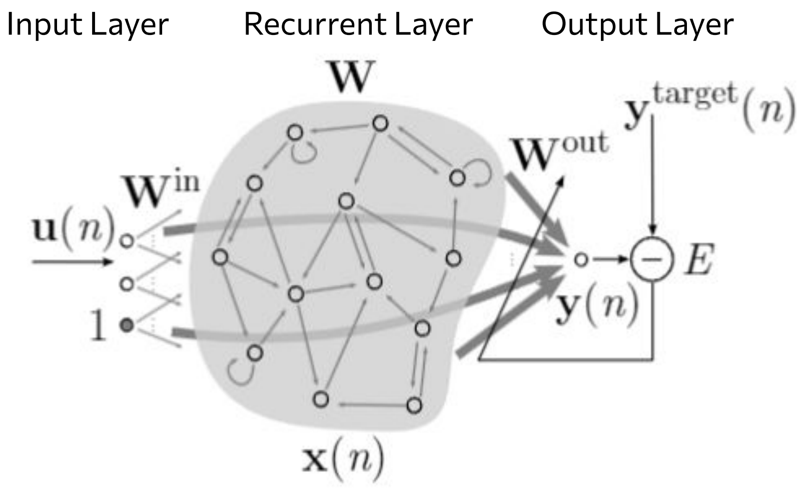



Fig. 1: An echo state network.

# Echo State Network

| | | |
|---|---|---|
| ρ(W)* | **Spectral Radius** | 0.1 |
| σ* | **Win Scaling St.Dev** | 0.5 |
| α* | **Leaking Rate** | 1.0 (default) |
| N | **Reservoir Size** | 72 + 72 |
| | **Training Length** | 500,000 |
| | **Prediction Length** | 1,000-10,000 |

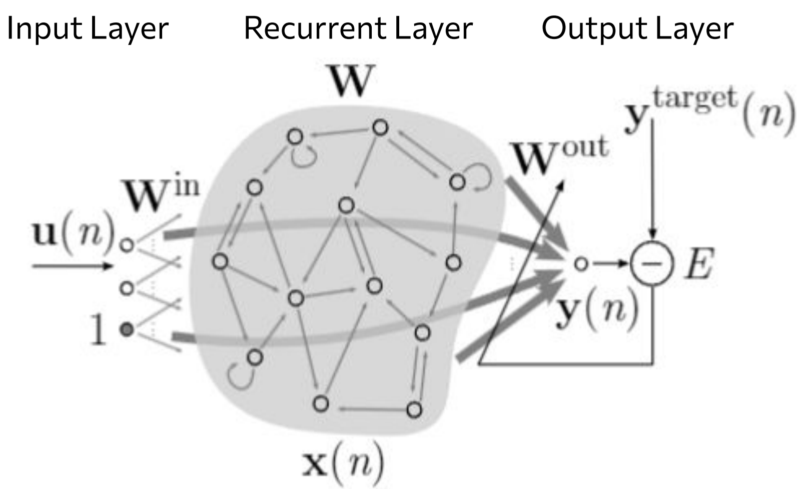
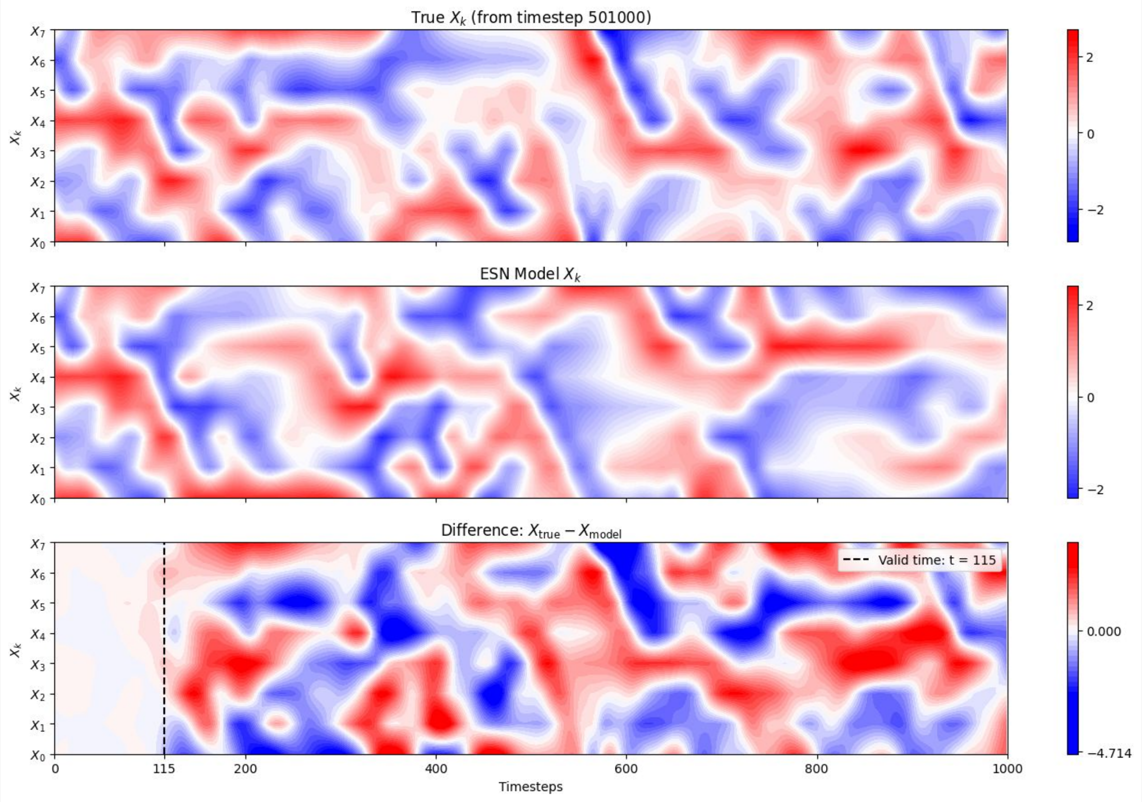
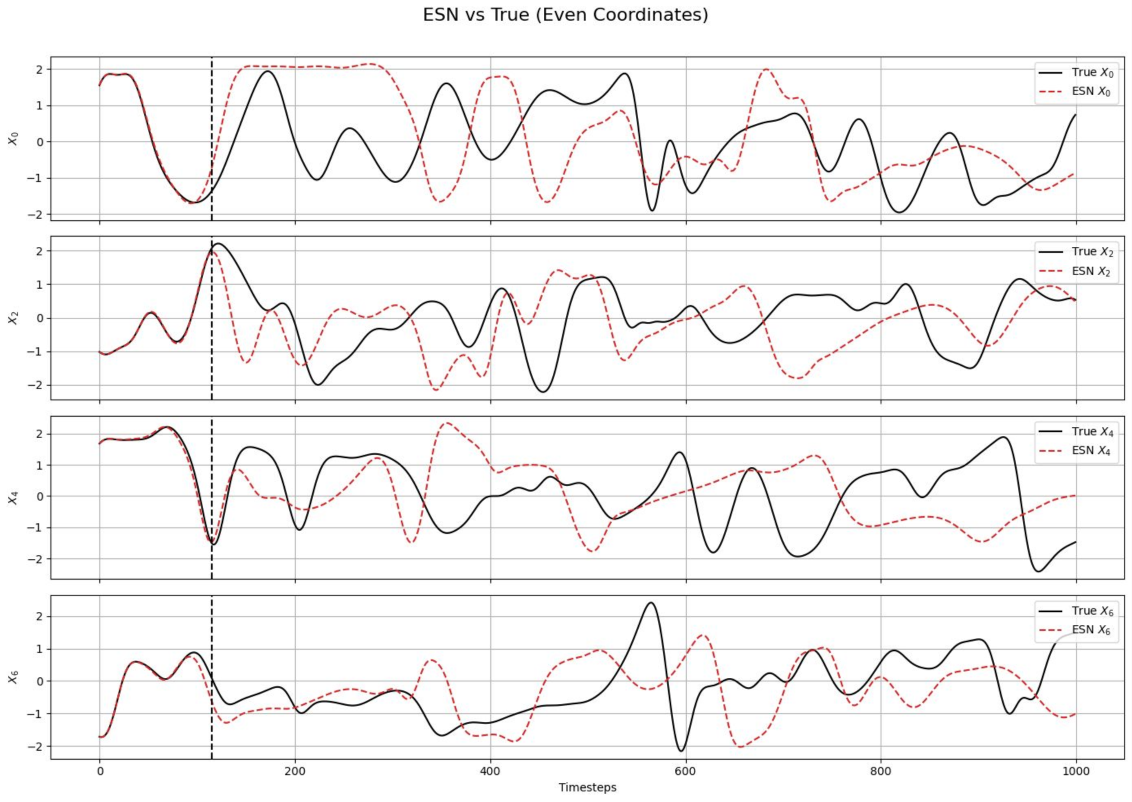



Fig. 1: An echo state network.

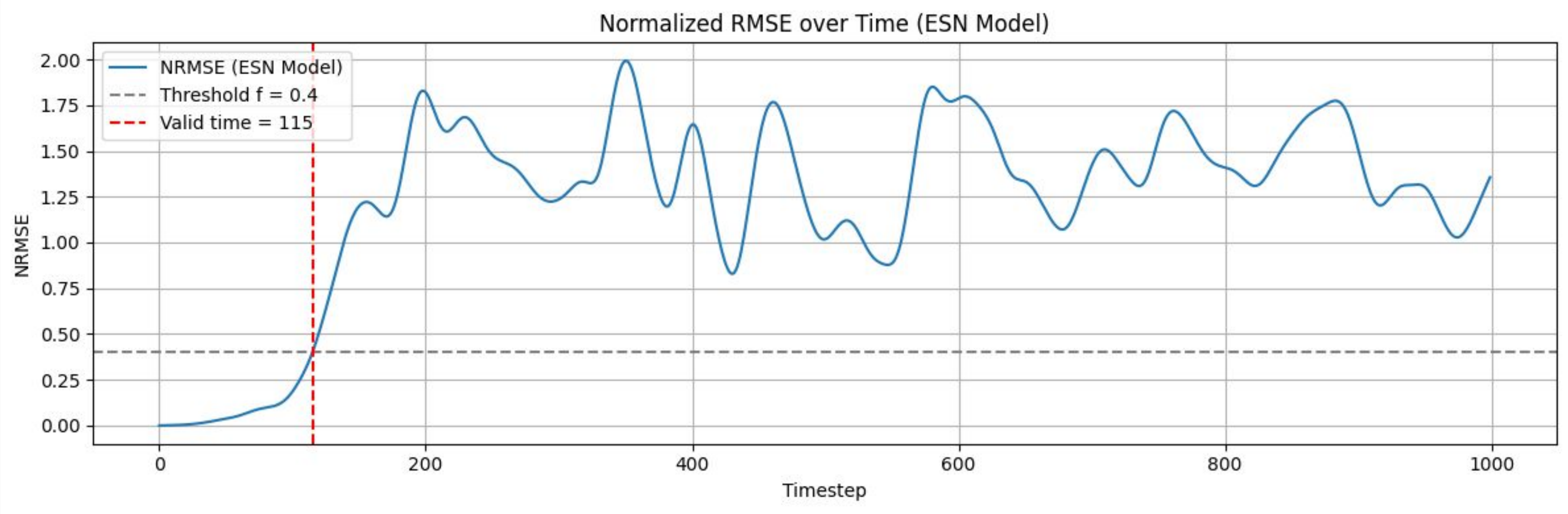
# Echo State Network



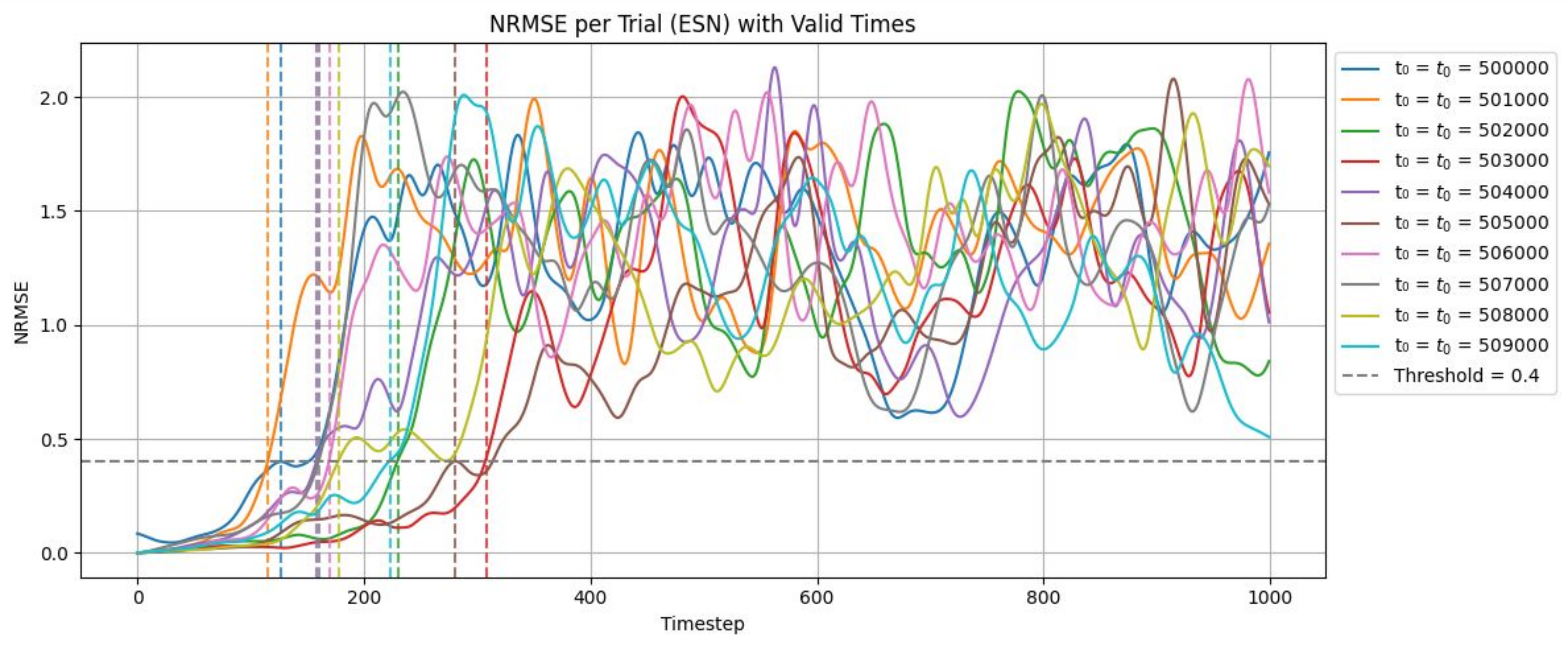
# Echo State Network



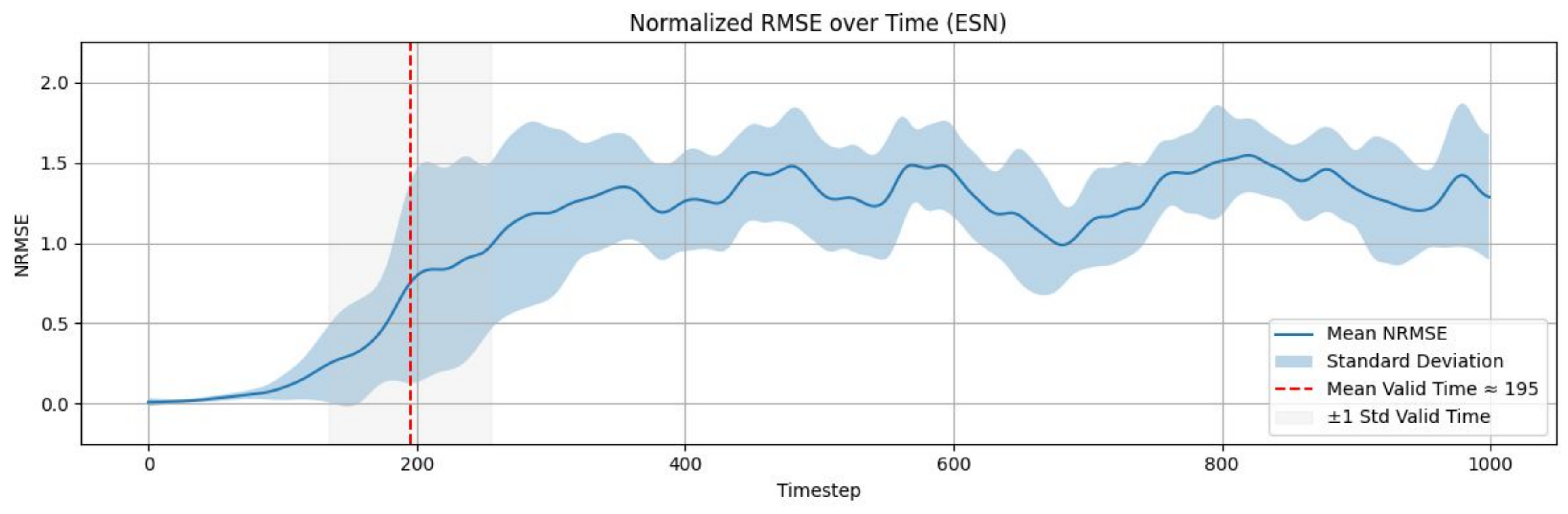
# Echo State Network



# Echo State Network



# Echo State Network



# Hybrid Echo State Network

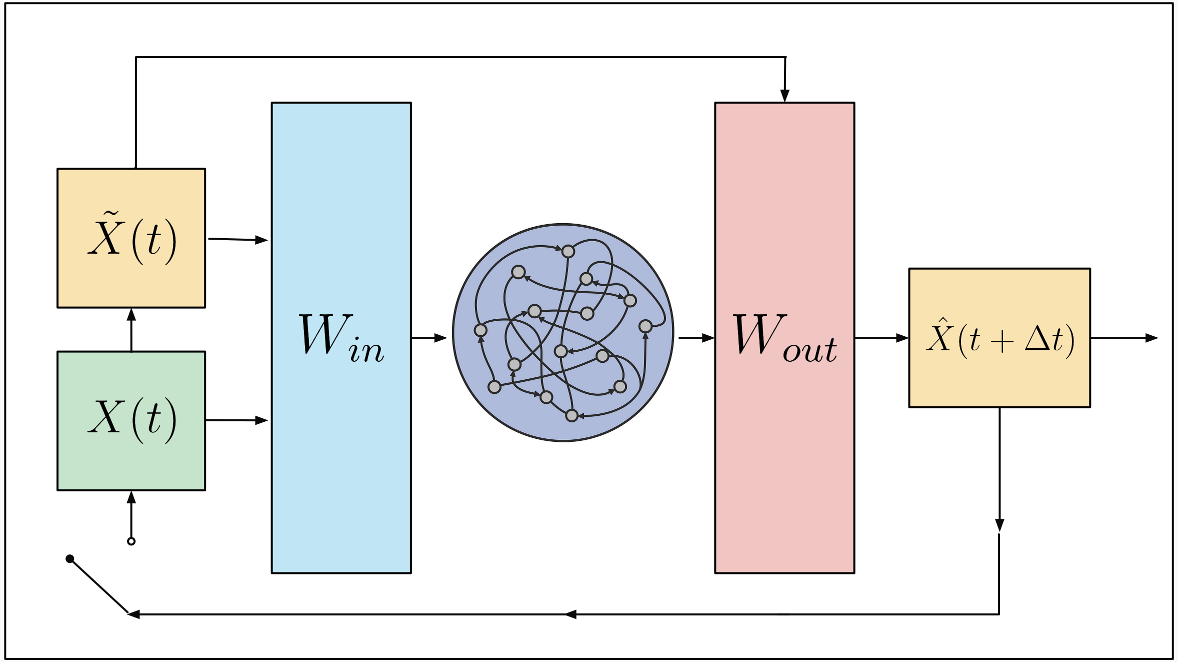
```
def train(r_states, X_model, X_true):
```

$$r_j^*(t) = \begin{cases} r_{j-1}(t) \cdot r_{j-2}(t), & \text{if } j \text{ is even} \\ r_j(t), & \text{otherwise} \end{cases}$$

$$u(t) = \begin{bmatrix} X_{\text{model}}(t) \\ r^*(t) \end{bmatrix} \in \mathbb{R}^{(K+N)\times 1}$$

$$Y = [X_{\text{true}}(t)]$$

$$W_{\text{out}} = YU^\top(UU^\top + \beta I)^{-1}$$



# Hybrid Echo State Network

```
def predict_hybrid(A, Win, W_out, r_state, X_init, Y_init, res_params, r_idx, start_idx)
```

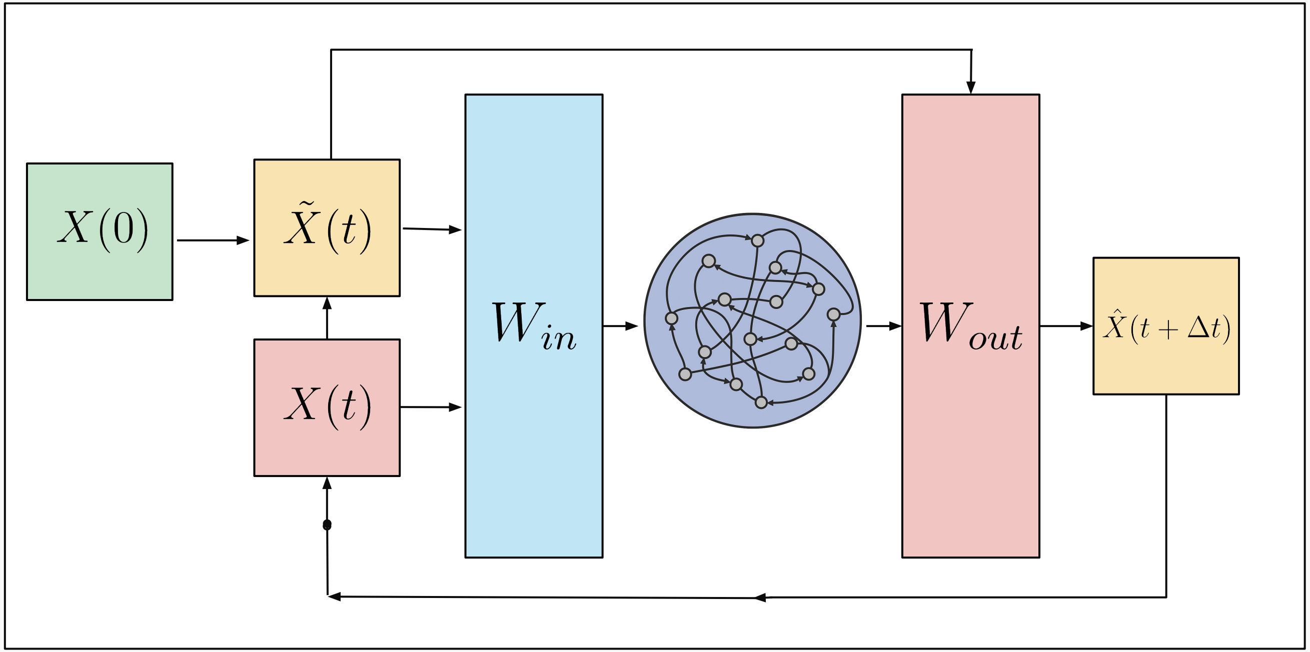
$$r_j^*(t) = \begin{cases} r_{j-1}(t) \cdot r_{j-2}(t), & \text{if } j \text{ is even} \\ r_j(t), & \text{otherwise} \end{cases}$$

$$U(t) = \begin{bmatrix} X_{\text{model}}(t) \\ r^*(t) \end{bmatrix}^T \in \mathbb{R}^{1\times(K+N)}$$

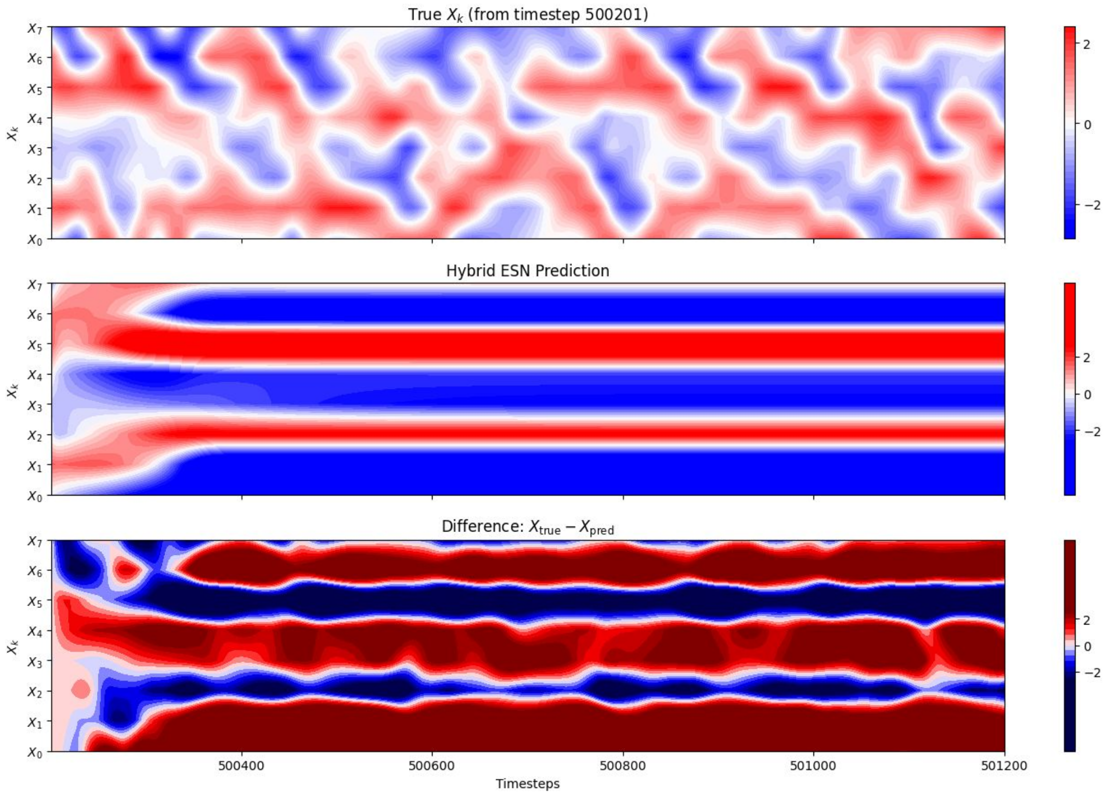
$$X_{\text{pred}}(t+\Delta t) = W_{\text{out}} \cdot \begin{bmatrix} X_{\text{model}}(t) \\ r^*(t) \end{bmatrix}^T$$

$$u(t) = \begin{bmatrix} X_{\text{model}}(t+\Delta t) \\ X_{\text{pred}}(t+\Delta t) \end{bmatrix} \in \mathbb{R}^{2K}$$

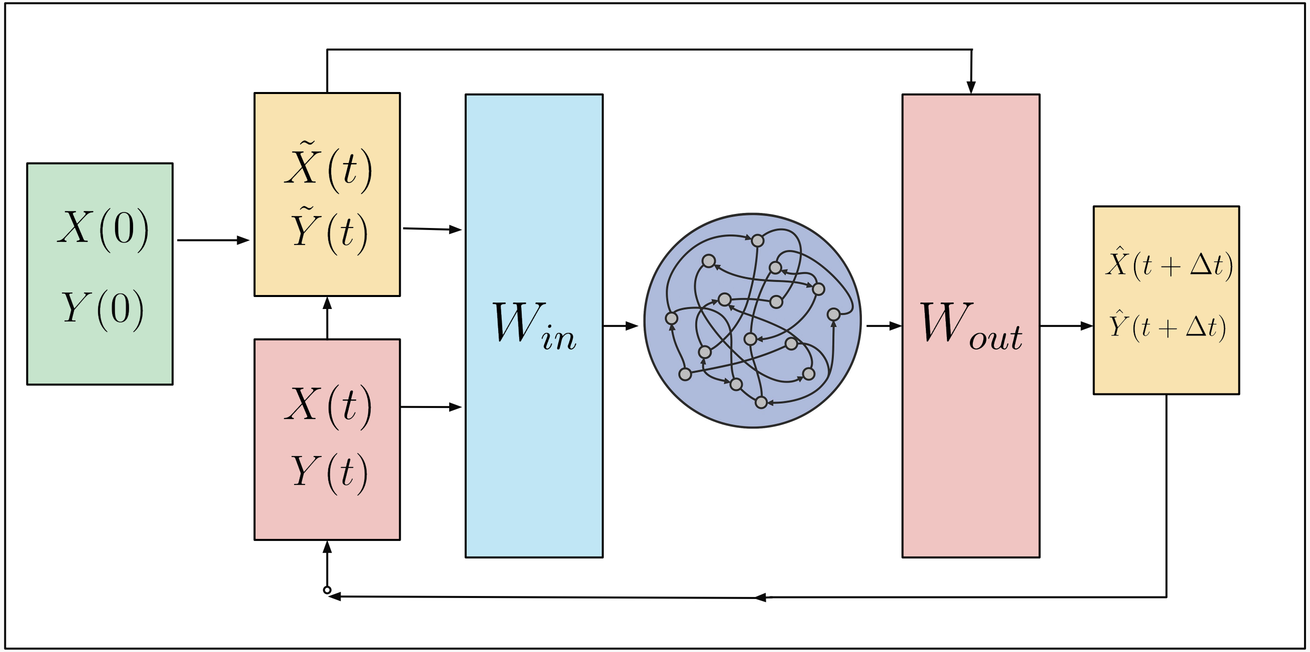
$$r(t+\Delta t) = \tanh\left(A \cdot r(t) + W_{\text{in}} \cdot u(t)\right)$$



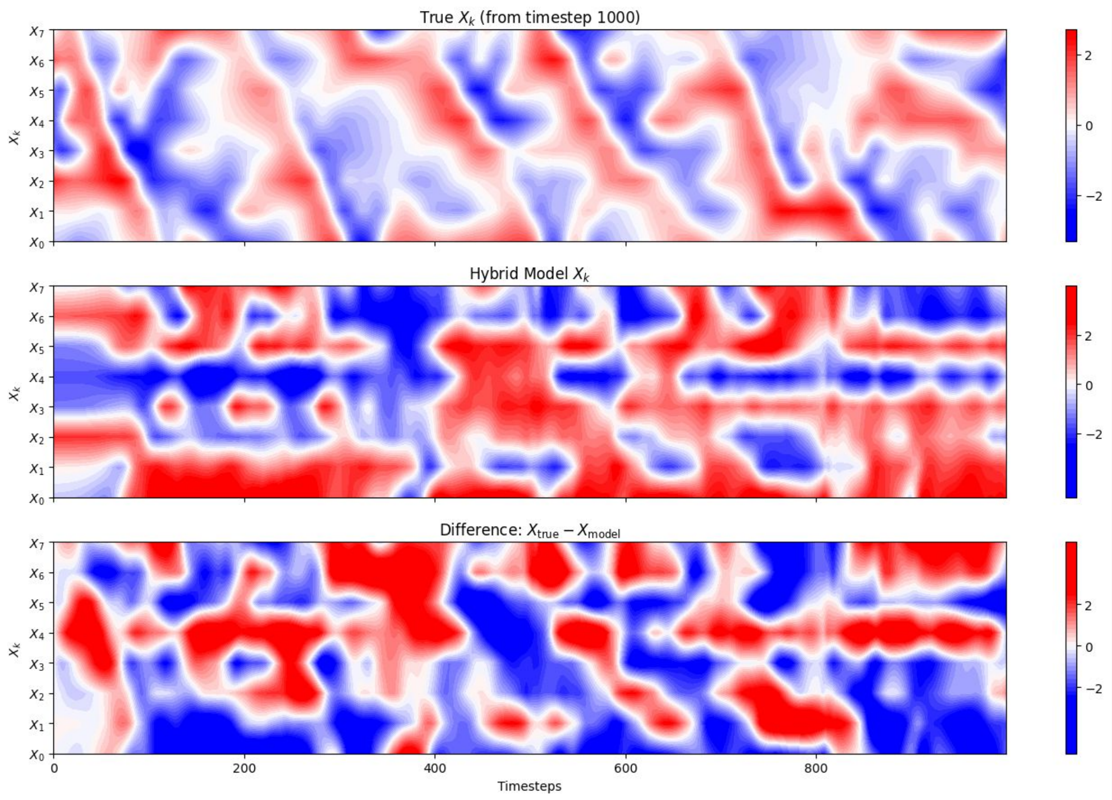
# Hybrid Echo State Network



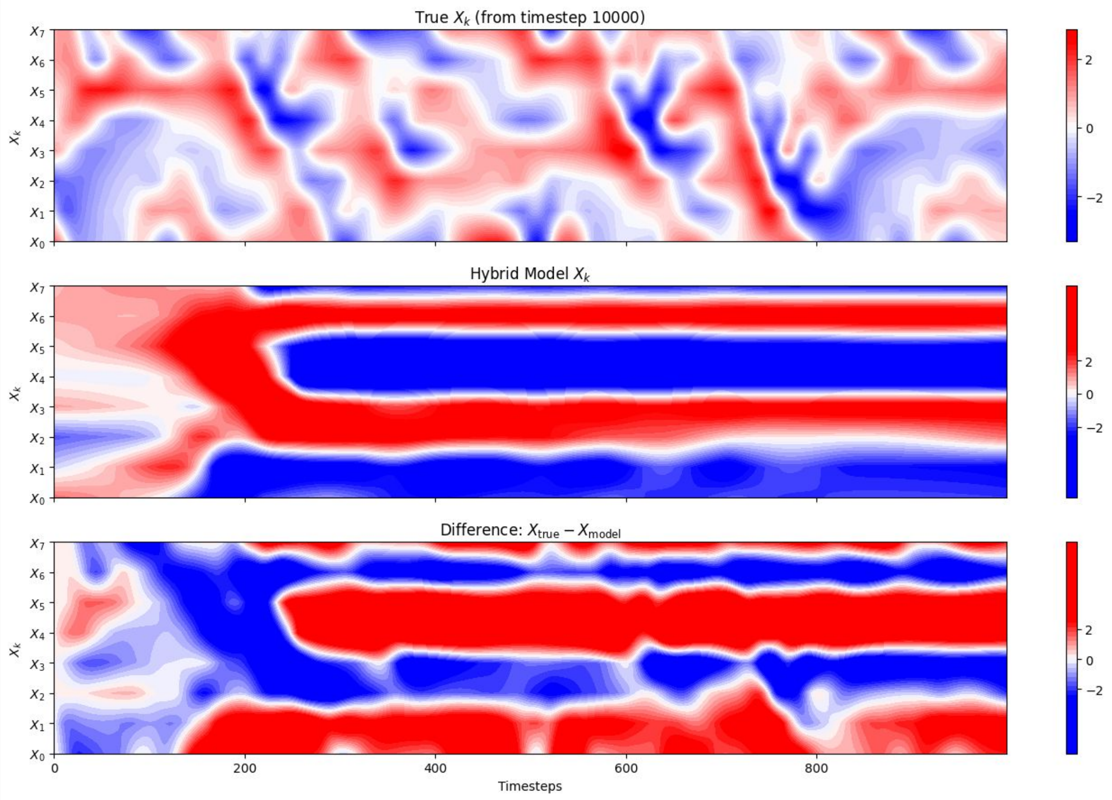
# Hybrid Echo State Network



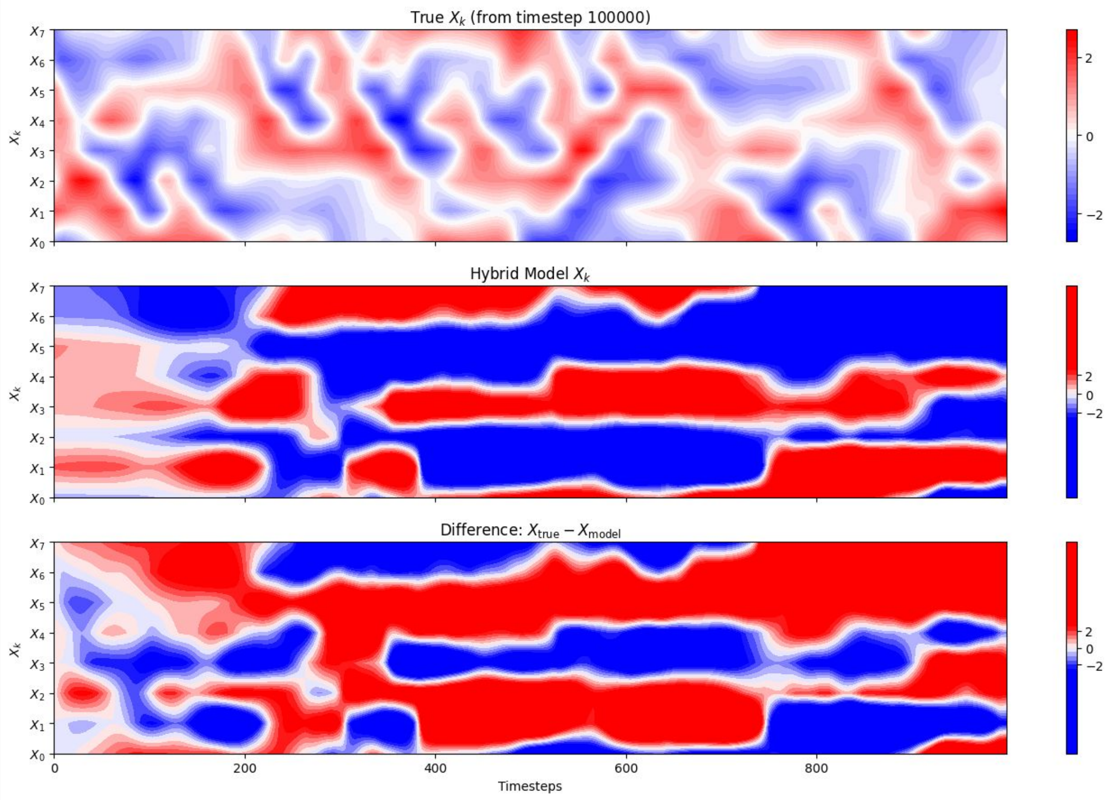
# Hybrid Echo State Network



# Hybrid Echo State Network



# Hybrid Echo State Network



# Ensemble Kalman Filter

- **Goal:** Estimate the true state of a system over time by combining model predictions with noisy observations.
- **Kalman Filter:** Minimizes uncertainty in state estimates.
- **ENKF:** Uses multiple simulations (ensemble members) with slight variations to approximate the best trajectory based on the observation.

# EnKF Implementation

- X = True system state vector
- f(x) = Forecast model (RK4 step)
- Y = Observation vector
- H = Observation matrix (full state observed)
- w = Process noise (added in predict())
- v = Measurement noise

State Evolution:

$$x_{t+1} = f(x_t) + w_t$$

Observations:

$$y_t = Hx_t + v_t$$

# Kalman Gain (K)

$$K = P_{xy}(P_{yy} + R)^{-1}$$

- Balances trust between the model and observations
- Pxy: Cross-covariance between ensemble state and predicted observation
- Pyy: Covariance of predicted observations
- R: Measurement noise covariance (R = np.eye(K) * 0.1)
    - Models the uncertainty in observations
    - The greater the values in R, the less the filter trusts the observations
- Large K is an indicator to trust observations
- Small K is an indicator to trust the model

# EnKF Class

```
def initialize_ensemble(self, x0, P0):
    self.ensemble = np.random.multivariate_normal(x0, P0, self.Ne)
```

- Initialize *Ne* ensemble members with an initial state centered at x0 and initial spread P0.

```
def predict(self, forecasted_x):
    for i in range(self.Ne):
        self.ensemble[i] = forecasted_x + np.random.normal(0, 0.5, forecasted_x.shape)
```

- Forecast each ensemble member
- Add Process noise to simulate uncertainty in dynamics

# EnKF Class

```python
def update(self, observation):
    X = self.ensemble.T
    x_mean = np.mean(X, axis=1, keepdims=True)
    X_prime = X - x_mean

    HX = self.H @ X
    y_mean = np.mean(HX, axis=1, keepdims=True)
    Y_prime = HX - y_mean

    P_xy = X_prime @ Y_prime.T / (self.Ne - 1)
    P_yy = Y_prime @ Y_prime.T / (self.Ne - 1) + self.R
    K = P_xy @ np.linalg.inv(P_yy)

    for i in range(self.Ne):
        perturb = np.random.multivariate_normal(np.zeros(self.R.shape[0]), self.R)
        innovation = observation + perturb - HX[:, i]
        X[:, i] += K @ innovation

    self.ensemble = X.T
    self.x_mean.append(np.mean(self.ensemble, axis=0))
    self.x_ens.append(np.copy(self.ensemble))
```

- Calculate the kalman gain using cross-covariance and predicted observation covariance
- For each ensemble member, create a random perturbation and compute the innovation
- Update ensemble member by nudging it in the direction of the innovation, scaled by the kalman gain

# Imperfect + EnKF Forecast

```
def imperfect_enkf_forecast(X_init, Y_init, observations, enkf, x_mean, x_std, y_mean, y_std, dt):
    """
    Propagate the imperfect model and use EnKF to correct every obs_every steps.
    """
    T = observations.shape[0]
    K = X_init.shape[0]
    X = X_init.copy()
    Y = Y_init.copy()

    X_preds = np.zeros((T, K))

    for t in trange(T):
        # Step the imperfect model
        X, Y = imperfect_model(X, Y, x_mean, x_std, y_mean, y_std, dt)

        # Pass the model forecast to EnKF
        enkf.predict(X)

        # If we have an observation, use it to correct X
        if not np.isnan(observations[t, 0]):
            enkf.update(observations[t])
            X = enkf.x_mean[-1]  # Replace X with filtered ensemble mean

        X_preds[t] = X  # Store the current corrected prediction

    enkf_mean, enkf_ens = enkf.get_results()
    return X_preds, enkf_mean, enkf_ens
```

- Use RK4 to step forward the model
- If there is an observation, apply the enkf update function

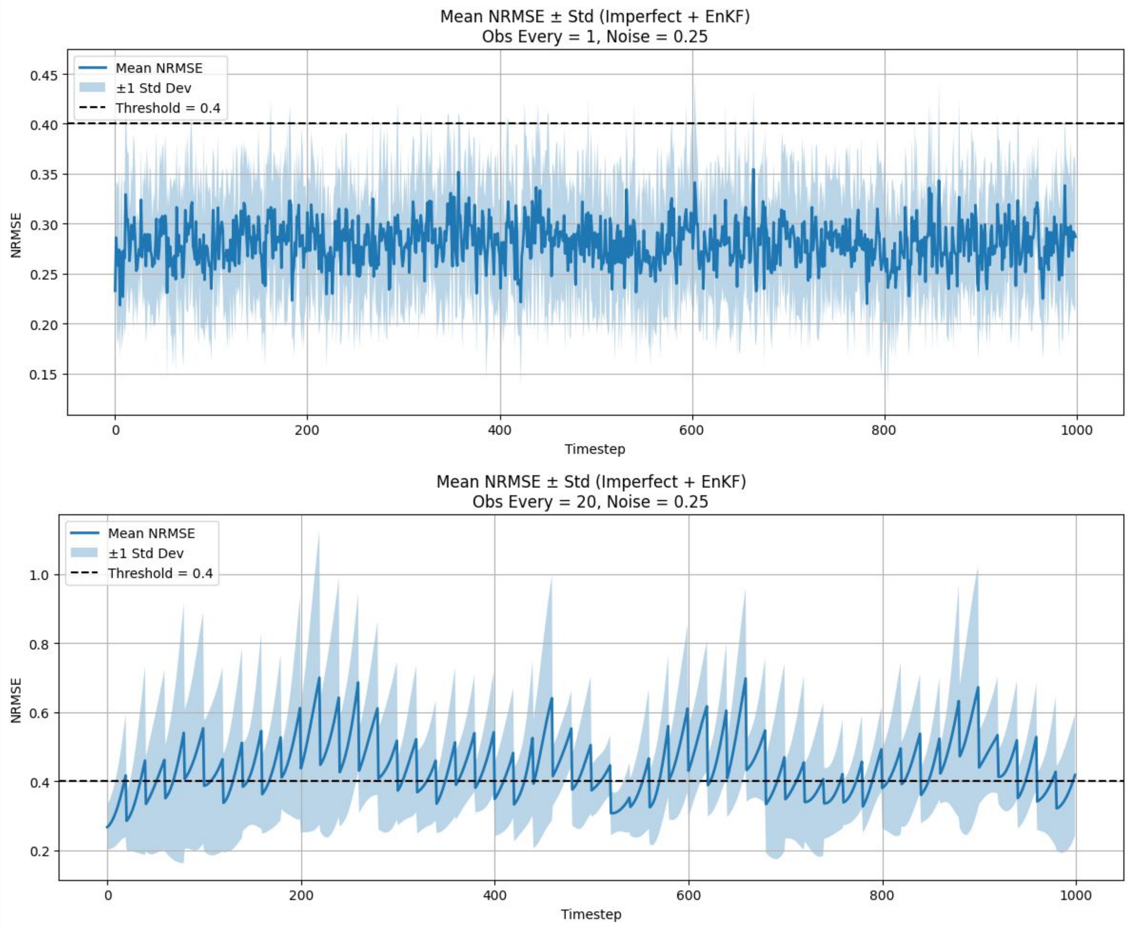
# EnKF Hyperparameters

We did a 4x4 grid testing 10 instances of:

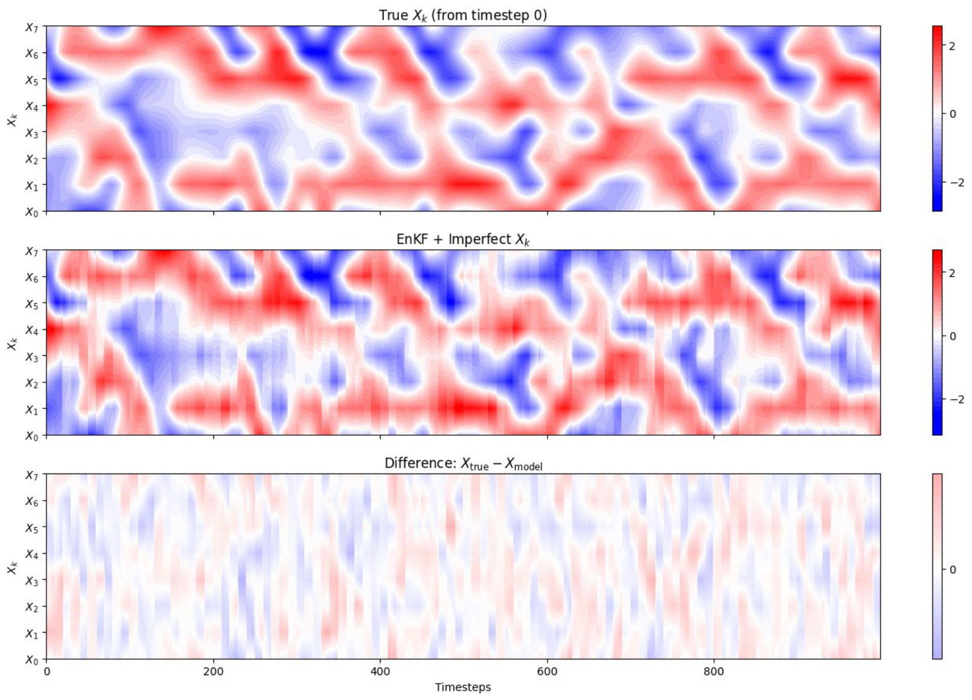
**Update Intervals**
[1,5,10,20] timesteps

**Noisy data variance**
[0.05,0.1,0.5,1] var.

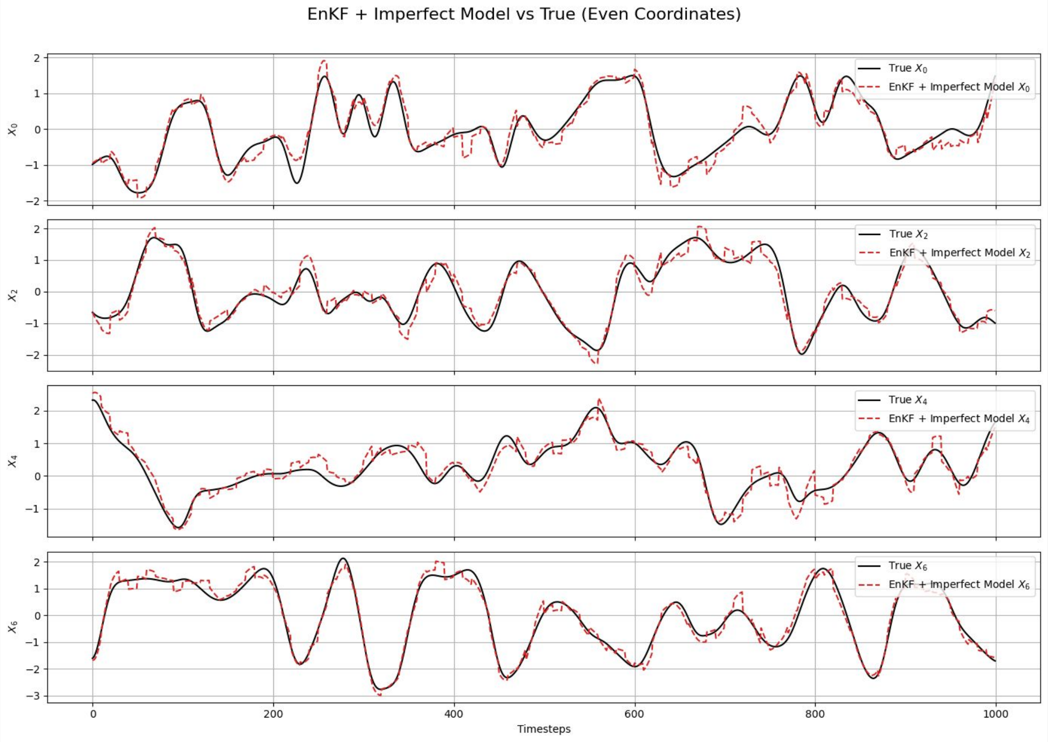
In accordance to other literature, we viewed an NRMSE threshold at 0.4.



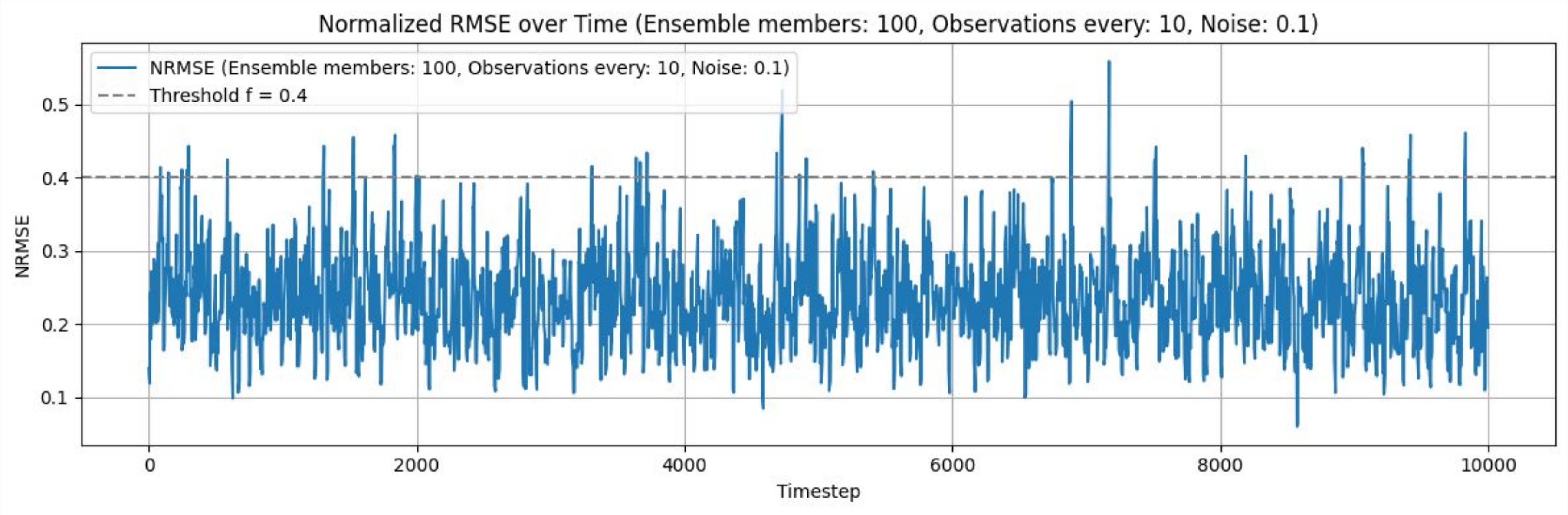
# Imperfect Model + EnKF



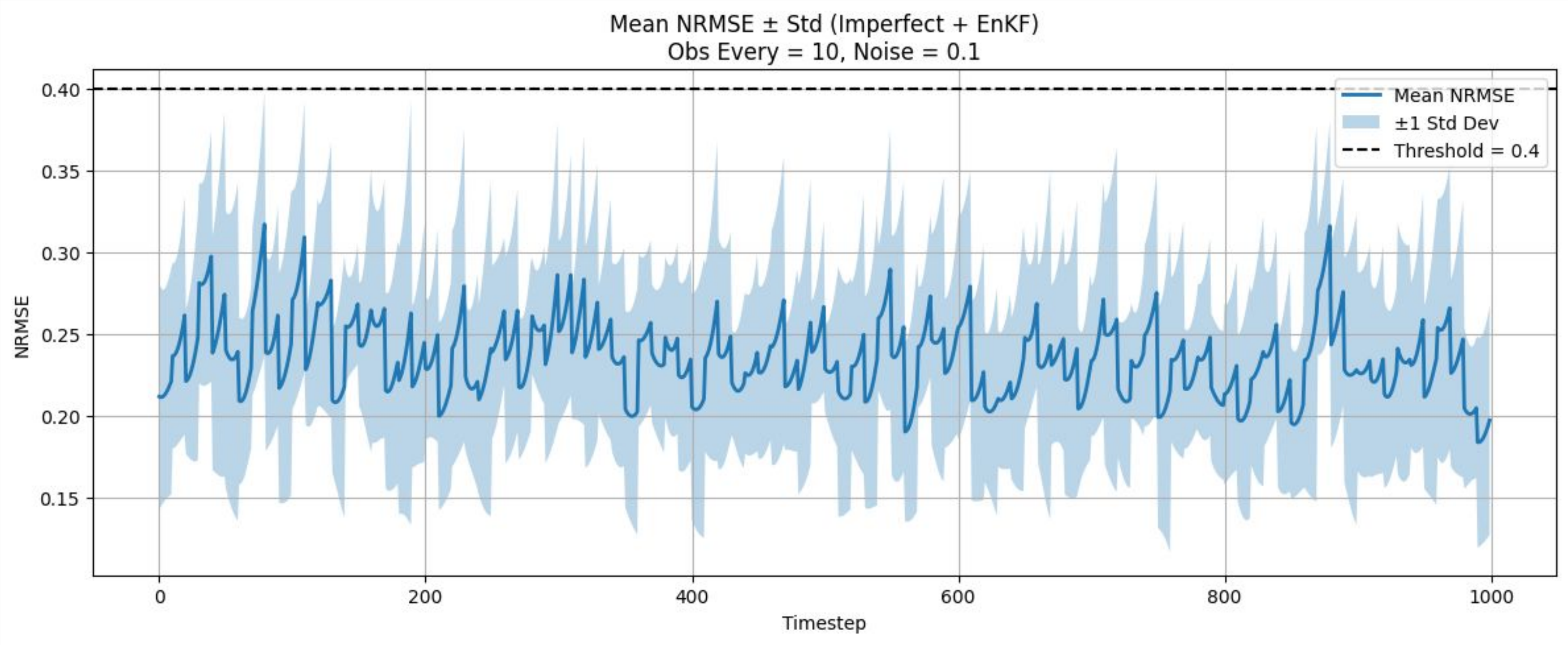
# Imperfect Model + EnKF



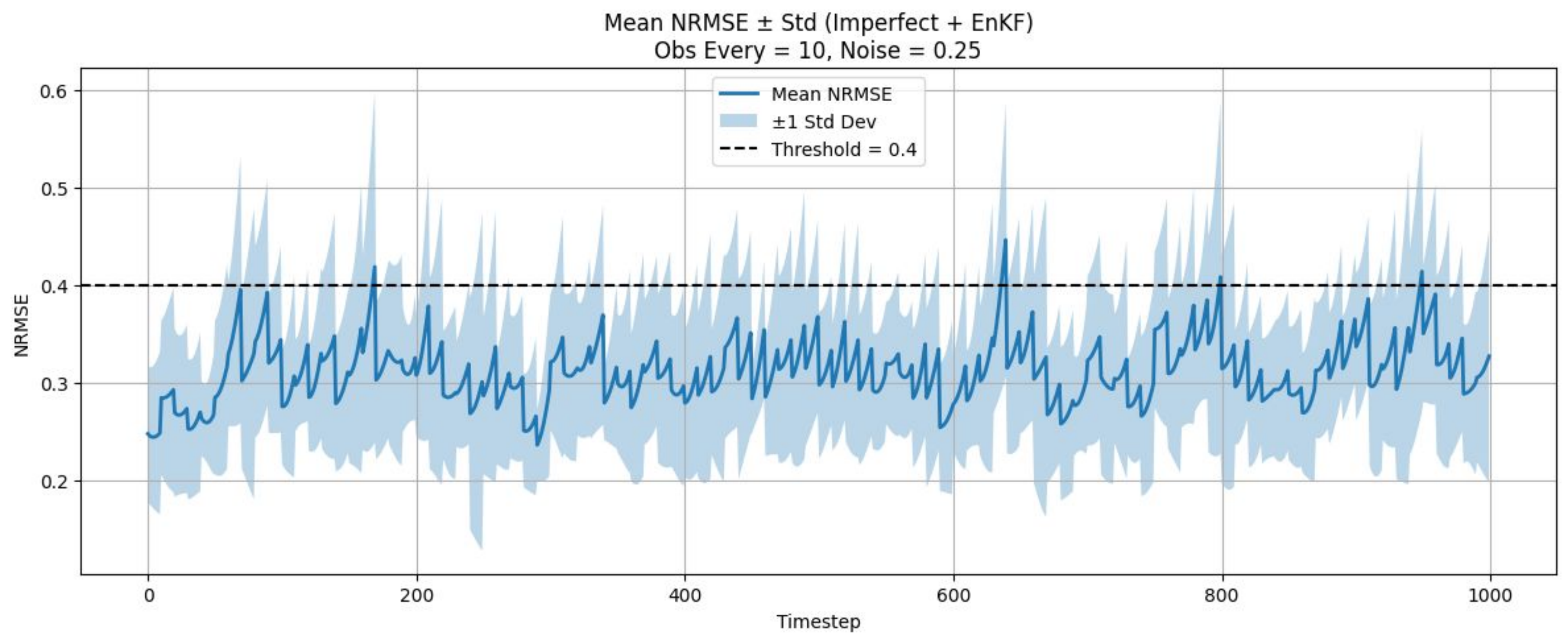
# Imperfect Model + EnKF



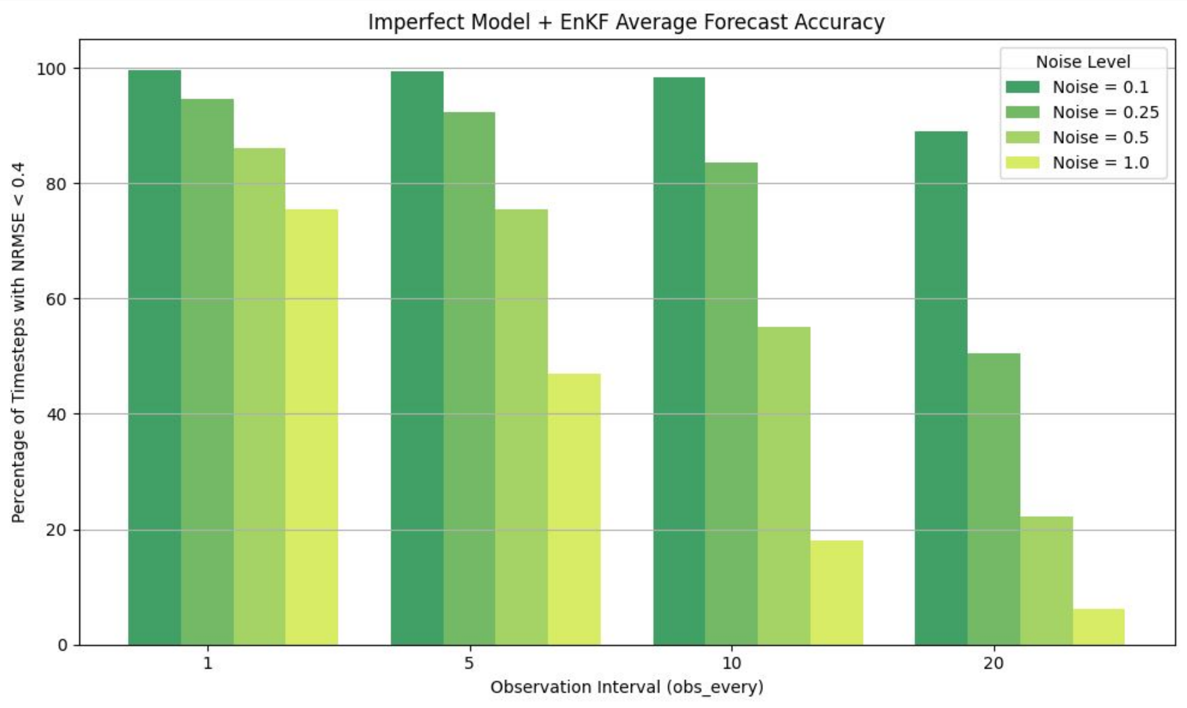
# Imperfect Model + EnKF



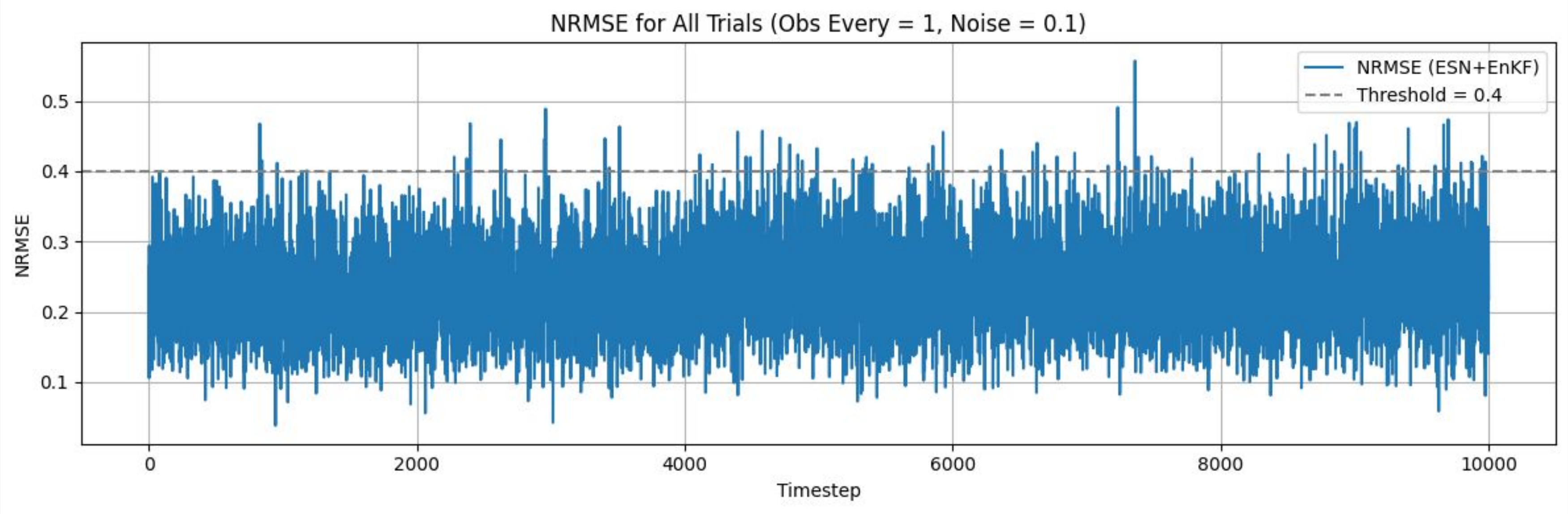
# Imperfect Model + EnKF



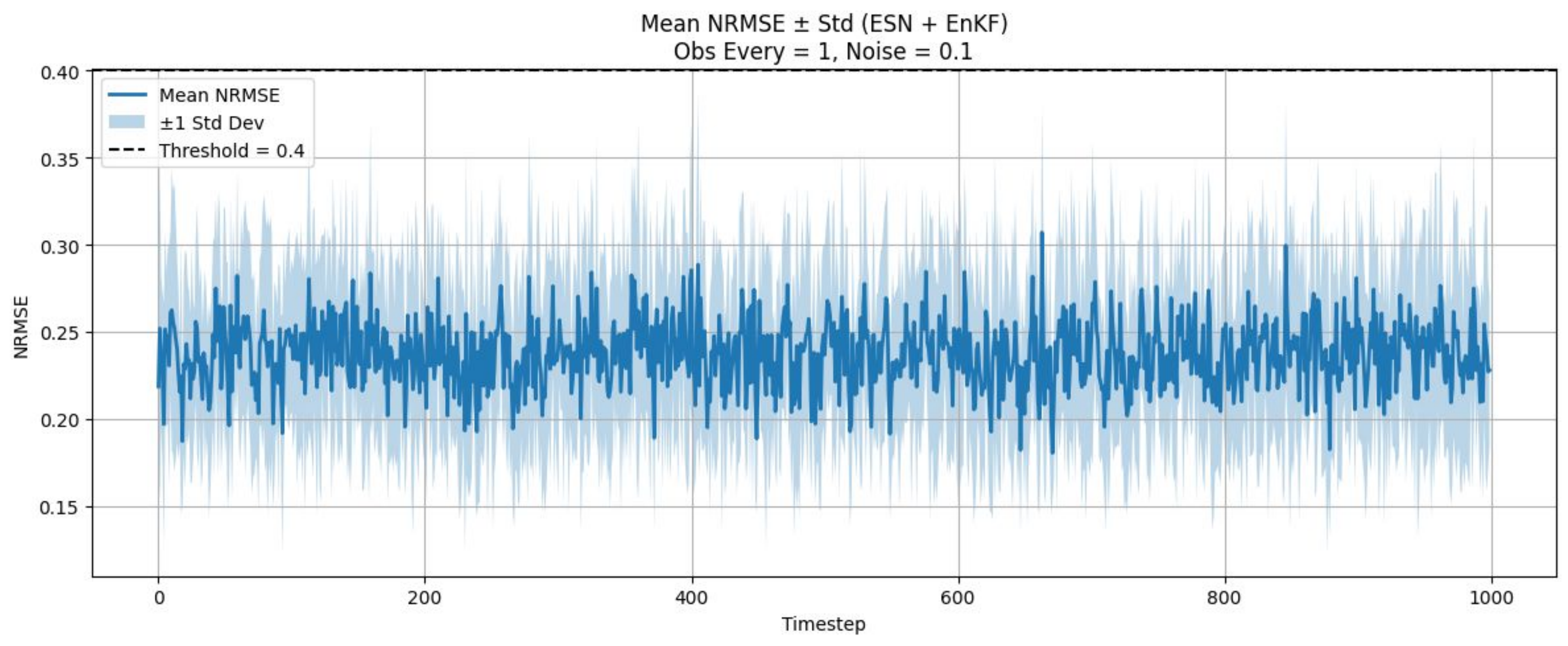
# Imperfect Model + EnKF



# ESN+ EnKF

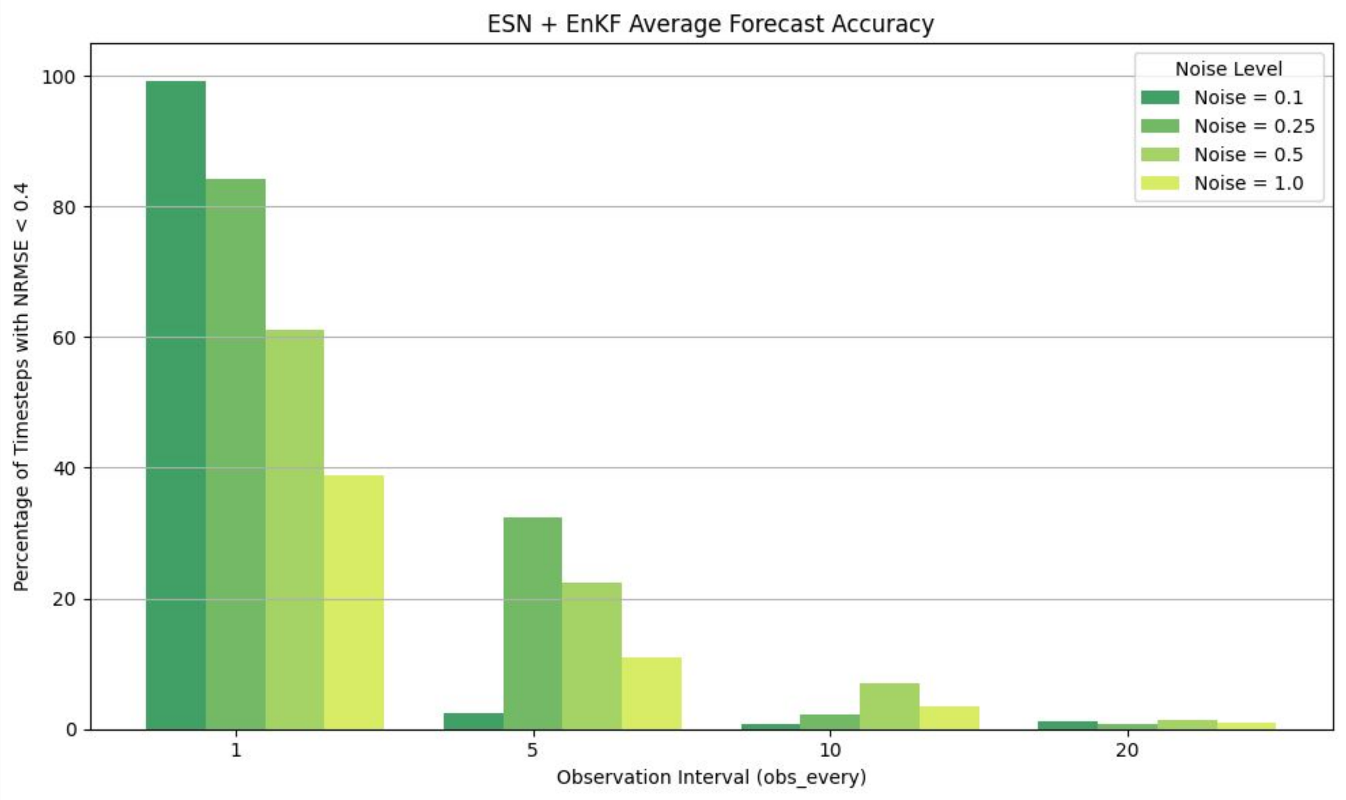

NRMSE for All Trials (Obs Every = 1, Noise = 0.1)
NRMSE (ESN+EnKF)
Threshold = 0.4
NRMSE
0.1
0.2
0.3
0.4
0.5
0
2000
4000
6000
8000
10000
Timestep

# ESN+ EnKF

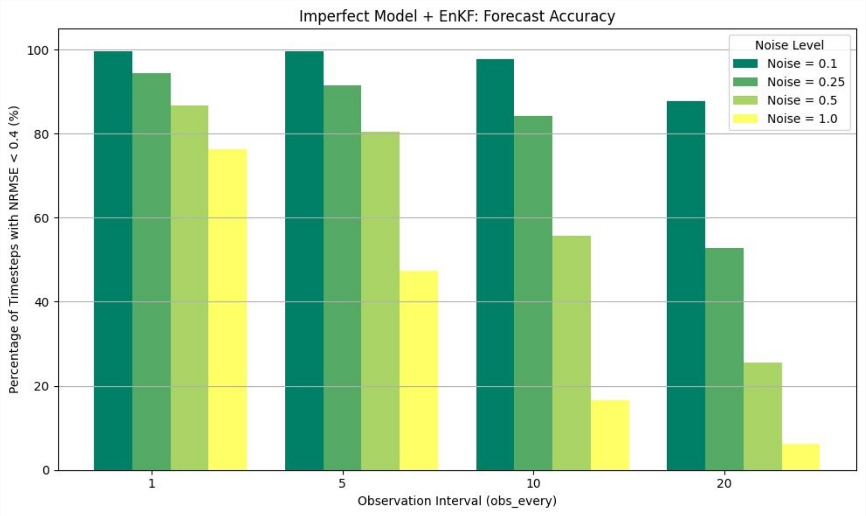


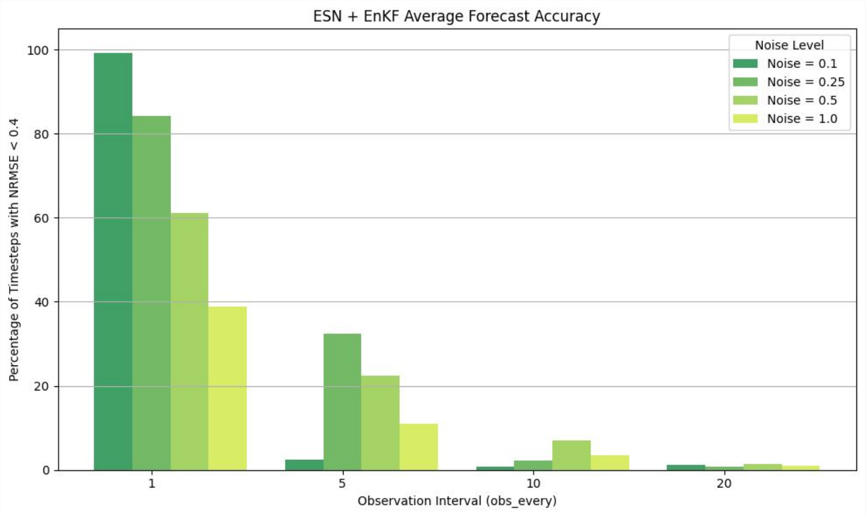
Mean NRMSE ± Std (ESN + EnKF)
Obs Every = 1, Noise = 0.1
Mean NRMSE
±1 Std Dev
Threshold = 0.4
NRMSE
Timestep
0.40
0.35
0.30
0.25
0.20
0.15
0
200
400
600
800
1000

# ESN + EnKF

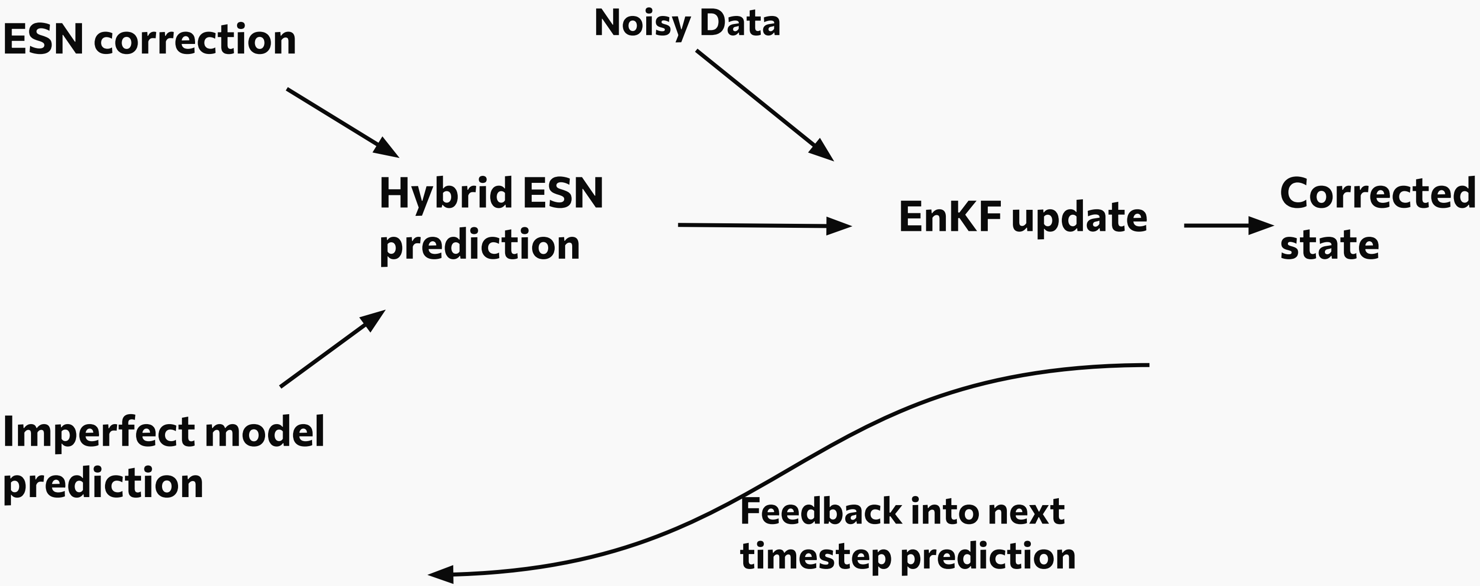

ESN + EnKF Average Forecast Accuracy
Percentage of Timesteps with NRMSE < 0.4
Observation Interval (obs_every)
Noise Level
Noise = 0.1
Noise = 0.25
Noise = 0.5
Noise = 1.0
0
20
40
60
80
100
1
5
10
20

# Imperfect Model vs. ESN + EnKF



# EnKF's Role in Hybrid ESN Model



# Task list

- Data generation and noising
  - 100% complete - Worked around GPU bottlenecks
- Hybrid ESN implementation
  - 70% - Got an ESN working, Hybrid Model showed numerical instability
- Ensemble Kalman Filter
  - 100% - Implemented and tested various parameters
- Training and Testing
  - 80% - Hybrid ESN Limitations
- Evaluating performance metrics
  - 100% - Wrote validation and metrics code

# Takeaways and Future Work

EnKF Trade-off in Data Reliability / Natural Noise

Understand Numerical Instability in the Hybrid ESN

Other PDE systems

# Thank You

**Questions?**