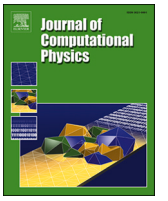
Journal of Computational Physics 547 (2026) 114550

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Learning enhanced ensemble filters

Eviatar Bach [a,b,c], Ricardo Baptista [d,e], Edoardo Calvello [f], Bohan Chen [f,*], Andrew Stuart [f]

[a] Department of Meteorology, University of Reading, Brian Hoskins Building, Reading, RG6 6ET, UK
[b] Department of Mathematics and Statistics, University of Reading, Pepper Lane, Reading, RG6 6AX, UK
[c] National Centre for Earth Observation, Brian Hoskins Building, University of Reading, Reading, RG6 6ET, UK
[d] Department of Statistical Sciences, University of Toronto, Ontario Power Building, 700 University Ave., Toronto, M5G 1Z5, ON, Canada
[e] Vector Institute for Artificial Intelligence, 661 University Ave., Toronto, M5G 1M1, ON, Canada
[f] The Computing + Mathematical Sciences Department, California Institute of Technology, 1200 E California Blvd, Pasadena, 91125, CA, USA

ARTICLE INFO

Keywords:
Data assimilation
Ensemble filter
Neural operators

ABSTRACT

The filtering distribution in hidden Markov models evolves according to the law of a mean-field model in state–observation space. The ensemble Kalman filter (EnKF) approximates this mean-field model with an ensemble of interacting particles, employing a Gaussian ansatz for the joint distribution of the state and observation at each observation time. These methods are robust, but the Gaussian ansatz limits accuracy. Here this shortcoming is addressed by using machine learning to map the joint predicted state and observation to the updated state estimate. The derivation of methods from a mean field formulation of the true filtering distribution suggests a single parametrization of the algorithm that can be deployed at different ensemble sizes. And we use a mean field formulation of the ensemble Kalman filter as an inductive bias for our architecture.

To develop this perspective, in which the mean-field limit of the algorithm and finite interacting ensemble particle approximations share a common set of parameters, a novel form of neural operator is introduced, taking probability distributions as input: a *measure neural mapping* (MNM). A MNM is used to design a novel approach to filtering, the *MNM-enhanced ensemble filter* (MNMEF), which is defined in both the mean-field limit and for interacting ensemble particle approximations. The ensemble approach uses empirical measures as input to the MNM and is implemented using the set transformer, which is invariant to ensemble permutation and allows for different ensemble sizes. In practice fine-tuning of a small number of parameters, for specific ensemble sizes, further enhances the accuracy of the scheme. The promise of the approach is demonstrated by its superior root-mean-square-error performance relative to leading methods in filtering the Lorenz '96 and Kuramoto-Sivashinsky models.

1. Introduction

Filtering, determining the conditional distribution of the state of a partially and noisily observed dynamical system given data collected sequentially in time, constitutes a longstanding computational challenge, particularly when the state is high-dimensional. Sequential data assimilation encompasses a broad range of methods aimed at finding the filtering distribution, or simply estimating the state, in an online fashion. The focus of this work is on the use of sequential data assimilation methods for filtering and state

* Corresponding author.
E-mail address: bhchen@caltech.edu (B. Chen).

https://doi.org/10.1016/j.jcp.2025.114550
Received 27 May 2025; Received in revised form 3 November 2025; Accepted 24 November 2025
Available online 4 December 2025
0021-9991/© 2025 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

estimation, and in particular the methodologies developed primarily in the geophysical sciences, and weather prediction especially, which are relevant in high dimensions. The overarching goal is to introduce a new framework for the *machine learning* of data assimilation (DA) algorithms, and specifically ensemble filters, which facilitates the sharing of parameters between implementations with different ensemble sizes; we may, for example, train and deploy the same model with different ensemble sizes. To achieve this we work in the context of mean-field state-space formulations of filtering. This perspective enables us to consider the idea of learning algorithms which share a common set of parameters, regardless of ensemble size, by viewing large ensembles as approximating a common mean field limit. In practice, small ensemble sizes behave differently from large ensemble sizes, an issue that has heretofore been addressed by use of inflation and localization. Here we show that an efficient fine-tuning strategy can be used to transfer parameters trained at one ensemble size to another ensemble size, at which the algorithm can be deployed. The resulting algorithms are based on stochastic dynamical systems for a state variable whose evolution depends on its own law; particle approximation leads to implementable algorithms. We show how the attention mechanism, adapted from the transformer methodology at the heart of large language models, can be used to define probability measure-dependent transport maps, i.e. mappings that move probability mass from one distribution to another, that can be approximated by empirical measures using arbitrary ensemble sizes. We refer to the resulting class of neural network architectures as *measure neural mappings*.

In Section 1.1, we set our work in the context of pre-existing literature. We introduce notation used throughout the paper in Section 1.2. Then, we describe the specific contributions we make to achieve our overarching goal, and we overview the contents of the paper in Section 1.3.

1.1. Literature review

Data assimilation (DA) is a fundamental framework for integrating observational data with numerical models for the purpose of state estimation and forecasting. DA methods systematically combine observations with prior model forecasts, accounting for their respective uncertainties, to produce optimal estimates of the system state or to characterize a probability distribution over the state. Several textbooks provide comprehensive overviews of this field [1–6].

The Kalman filter [7] is a foundational sequential data assimilation algorithm for linear systems with Gaussian errors. The extended Kalman filter, using linearization, adapts the Kalman methodology to nonlinear systems, but its direct application to high-dimensional geophysical problems is computationally prohibitive. The ensemble Kalman filter (EnKF) [8–10] addresses this limitation through a Monte Carlo approach, representing statistics via a limited ensemble of model states. Square-root variants, including the ensemble transform Kalman filter and the ensemble adjustment Kalman filter [10,11], reduce both storage requirements and sampling errors through deterministic formulations. Stochastic EnKF variants update each ensemble member using the Kalman filter equation with random perturbations in the innovation[12]; provides a consistent derivation that favors perturbing the modeled observations. However, EnKF-based methods commonly encounter challenges in practical implementations and, in particular, suffer from sampling errors caused by finite ensemble sizes. To address these limitations, covariance inflation [13,14] and localization [15] techniques have been developed. Approaches such as the iterative EnKF [16] and the maximum likelihood ensemble filter [17] further improve performance in nonlinear regimes.

The integration of machine learning methodologies into DA has recently emerged as a powerful strategy to enhance predictive accuracy and scalability beyond traditional approaches constrained by Gaussian assumptions. Such machine learning approaches are reviewed in [6,18]. A key direction has been the direct learning of filters from data. Learning a fixed-gain filter was considered in [19–22]. McCabe et al. [23] developed an ensemble filter that uses the mean and covariance matrix as input to a recurrent neural network. Bocquet et al. [24] further demonstrated the efficacy of neural network-based filtering schemes without relying on an ensemble, achieving comparable accuracy to ensemble methods by identifying key dynamical perturbations directly from forecast states.

Complementary to these direct learning strategies, a distinct class of approaches formulates ensemble filtering as a transport problem. These methods explicitly construct transformations that map forecast distributions onto analysis distributions. Representative techniques include Knothe–Rosenblatt rearrangements [25], normalizing flows [26], bridge matching [27], and optimal transport frameworks [28]. These methods offer rigorous probabilistic interpretations and enable accurate sampling from the filtering distribution at each assimilation step. Typically these approaches find transformations acting on each individual particle, rather than operating directly at the distributional level, and are not designed to directly transform or update the ensemble members collectively.

Beyond per-particle transformations, a complementary line of research updates the ensemble collectively via particle flows [29], with applications to atmospheric models [30]. Early particle-flow methods cast the analysis as a continuous ordinary differential equation that drives particles toward the posterior [31], with related continuous-time ensemble formulations such as the EnKF–Bucy perspective [32]. Variational or Stein-based mappings are considered to push forward the prior through learned maps, e.g., the variational mapping particle filter [33].

Probabilistic formulations provide another promising direction for learning-based filters. These approaches aim to directly approximate the Bayesian inference problem of obtaining the filtering distribution [6,22,34,35]. In particular, recent works have proposed frameworks to jointly learn the forecast and analysis steps [35], or assume knowledge of the dynamics and learn parameterized analysis maps via variational inference [22].

Building on the use of deep learning architectures for ensemble filtering, recent efforts have incorporated permutation-invariant neural networks to respect the unordered nature of ensembles. Such architectures enable models to learn interactions among ensemble members without being sensitive to their ordering, which is essential for consistent filter behavior across runs. These designs have been applied to directly learn ensemble update rules [36] and to refine ensemble forecasts in post-processing settings [37].

Transformers and the underlying attention mechanism [38] are now ubiquitous in machine learning. Their success at modeling global, long-range correlations has made this architecture an attractive methodology for operator learning; see for example [39–42]. In this article we are interested in the attention mechanism as a map between metric spaces of probability measures, thus going beyond the definition on Euclidean spaces from [38] and the generalization to Banach spaces in [41]. Recent developments in the mathematical analysis of transformers have led to the formulation of the continuum limit of self-attention layers [43]. In [43] the authors describe the evolution under this continuum limit architecture as the solution of a continuity equation; see [44] for an overview on this perspective. This formulation is also employed in [45] to analyze self-attention dynamics as a measure-to-measure map. However, this analysis is restricted to measures defined on the sphere and does not include cross-attention. In this article we present a general methodological framework for neural network architectures on the metric space of probability measures and a general definition of attention as a measure-to-measure map. We will leverage this formulation to build implementable methodologies effecting data assimilation.

1.2. Notation

Throughout the paper we use $\mathbb{N} = \{1, 2, \dots\}$, $\mathbb{Z}^+ = \{0, 1, 2, \dots\}$, $\mathbb{R} = (-\infty, \infty)$, and $\mathbb{R}^+ = [0, \infty)$ to denote the set of positive integers, non-negative integers, real numbers, and non-negative real numbers, respectively. We define $[N] = \{1, 2, \dots, N\}$. We use $\mathcal{F}(A)$ to denote the set of all nonempty finite subsets of a set A.

We denote by $\mathcal{P}(\Omega)$ the space of probability measures defined on set Ω. Given a sigma-algebra on Ω, we define a probability measure $\mathbb{P}$ on Ω and we let $\mathbb{E}$ denote expectation under this probability measure. We denote by $\mathrm{Law}(v)$ the law of a random variable v, and denote by $T_\#\pi$ the pushforward measure of π by the map T: if $u \sim \pi$ then $T(u) \sim T_\#\pi$. For $\tau \in \mathbb{R}^d$, we let δ_τ denote the Dirac mass on $\mathbb{R}^d$ centered at point τ. We use the notation $\mathsf{N}(m, C)$ for the Gaussian distribution with mean m and covariance C. We use the font mathsf for operators acting on the space of probability measures or the space of functions; we also use this convention for operations on the space of vector-valued sequences over $[N]$, denoted by $\mathcal{U}([N]; \mathbb{R}^d)$. If an operator B on probability measures is defined as the pushforward by a map, we denote the associated transport map with the font $\mathfrak{mathfrak}$, i.e. $\mathsf{B}(\cdot) = \mathfrak{B}_\#(\cdot)$.

In the remainder of this paper we assume that all probability measures are absolutely continuous with respect to the Lebesgue measure so that they have probability density functions, or that they comprise a convex combination of Dirac masses. We will refer to probability measures and probability density functions interchangeably, depending on the setting.

In the context of DA, we use v to denote the system states and y to denote the observations. The superscript $\dagger$ indicates true values and the subscript $j \in \mathbb{Z}^+$ denotes the time step. For ensemble methods, we use the parenthesized superscript (n) for the ensemble index.

1.3. Contributions and overview

Our five primary contributions are as follows:

1. We present a framework for learning ensemble filtering algorithms, based on a mean-field state-space formulation of the evolution of the filtering distribution.
2. We introduce a novel generalization of neural operators to maps defined to act between metric spaces of probability measures. We call such maps *measure neural mappings* (MNM). We identify an architecture to implement such maps, based on the attention mechanism.
3. The MNM architecture is used to define approximations of the mean-field formulation of the filtering distribution. This leads to a novel learning-based framework for ensemble methods: the *measure neural mapping enhanced ensemble filter* (MNMEF).
4. Because of the mean-field underpinnings, the basic learned parameters are shared between filters with different ensemble sizes. As a consequence, parameterizations learned at one ensemble size can be deployed at different ensemble sizes, leading to efficient training at small ensemble sizes. To enhance this approach, we introduce a fine-tuning methodology to incorporate a small number of parameters, such as localization and inflation, which intrinsically depend on ensemble size, into the resulting filters.
5. We demonstrate that the resulting filtering algorithm outperforms an optimized local ensemble transform Kalman filter (LETKF, a leading ensemble filter) on Lorenz '96, Kuramoto–Sivashinsky, and Lorenz '63 models, for both small and larger ensemble sizes.

In Section 2 we describe filtering from the perspective of mean-field dynamics and define a learning framework to approximate the true filter, addressing Contribution 1. Section 3 is devoted to the introduction of *measure neural mappings*, a generalization of neural operators to maps taking probability measures as inputs; we show in this section that the attention mechanism can be used to build a transformer architecture defined on probability measures, thereby tackling Contribution 2. Contributions 3 and 4 are addressed in Section 4, where we detail the algorithm for finite particle sizes. The numerical experiments in Section 5 support Contribution 5. We conclude in Section 6. We also note that a reader focused on methodological contributions alone can skip the self-contained Section 3.

2. Learning mean-field filters

In this section, we formulate the filtering problem (Section 2.1), describing sequential DA from the perspective of nonlinear evolution of probability densities (Section 2.2) and from the mean-field state-space evolution perspective (Section 2.3). We then present the general idea of learning an analysis map which has a probability measure as an input (Section 2.4). Finally, we introduce our

learning-based mean-field filter approach, which enhances traditional ensemble Kalman filtering by incorporating trainable correction terms in the mean-field equations (Section 2.5). This allows us to extend and, as will show in numerical results presented in the penultimate section, improve upon the standard EnKF methodology, which relies on a Gaussian ansatz; in so-doing we maintain the interpretable structure of Kalman filtering.

2.1. Filtering set-up

Consider the stochastic dynamics model given by

$$v_{j+1}^\dagger = \Psi(v_j^\dagger) + \xi_j^\dagger, \quad j \in \mathbb{Z}^+, \tag{2.1a}$$

$$v_0^\dagger \sim \mathsf{N}(m_0, C_0), \quad \xi_j^\dagger \sim \mathsf{N}(0, \Sigma) \text{ i.i.d.}, \tag{2.1b}$$

where $\Psi : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is the forward dynamic model, and $v_j^\dagger, \xi_j^\dagger \in \mathbb{R}^{d_v}$ for $j \in \mathbb{Z}^+$. We assume that the sequence $\{\xi_j^\dagger\}_{j\in\mathbb{Z}^+}$ is independent of initial condition $v_0^\dagger$; this is often written as $\{\xi_j^\dagger\}_{j\in\mathbb{Z}^+} \perp v_0^\dagger$. The data model is given by

$$y_{j+1}^\dagger = h(v_{j+1}^\dagger) + \eta_{j+1}^\dagger, \quad j \in \mathbb{Z}^+, \tag{2.2a}$$

$$\eta_j^\dagger \sim \mathsf{N}(0, \Gamma) \text{ i.i.d.}, \tag{2.2b}$$

where $h : \mathbb{R}^{d_v} \to \mathbb{R}^{d_y}$ is the observation operator and $y_j^\dagger, \eta_j^\dagger \in \mathbb{R}^{d_y}$ for $j \in \mathbb{Z}^+$. We assume that the sequence $\{\eta_j^\dagger\}_{j\in\mathbb{N}} \perp v_0^\dagger$ and that the two sequences in the dynamics and data models are independent of one another: $\{\xi_j^\dagger\}_{j\in\mathbb{Z}^+} \perp \{\eta_j^\dagger\}_{j\in\mathbb{N}}$. The states $v_j^\dagger$ lie in $\mathbb{R}^{d_v}$, while the observations $y_j^\dagger$ lie in $\mathbb{R}^{d_y}$. Note that there is a hidden parameter Δt specifying the time interval between steps j and $j+1$. It is implicitly embedded in (2.1) through Ψ, which represents the discrete-time map obtained by integrating the continuous dynamics over an interval of length Δt.

Remark 1. We use † for all variables defining the state and observation, and the noises that define them. This is to distinguish them from other similar variables, appearing without †, in the algorithms that follow. In this paper, we adopt autonomous dynamics with additive Gaussian noises according to (2.1) and (2.2). There are several extended settings, briefly outlined in Remark 2. These extensions are important in applications but introduce substantial technical challenges. We do not consider those settings in this paper for clarity of exposition.

Remark 2 (Possible extensions beyond the baseline model)**.** Beyond the autonomous, additive-Gaussian setting in (2.1)–(2.2), one may consider [4,46,47]:

- **Time-dependent dynamics/observations.** Allow Ψ, Σ, h, and Γ to vary with j. This can destroy stationarity and leads to non-homogeneous transition/likelihood kernels.
- **Multiplicative or non-Gaussian noises.** Replace additive Gaussian errors by state-dependent or heavy-tailed noises. This can induce heteroscedastic innovations and non-quadratic likelihoods.
- **Infinite-dimensional state spaces.** Model v_j in a function space (e.g., a separable Hilbert space) for PDE-governed systems. One must address well-posedness (e.g., trace-class noise/covariances), operator-valued updates, and consistency under spatial discretization/mesh refinement.

There are two primary types of problems in data assimilation: the filtering problem and the smoothing problem. In this paper, we focus on the filtering problem. To this end we define the accumulated data up to time j by $Y_j^\dagger := \{y_1^\dagger, \dots, y_j^\dagger\}$. Using this notation we state the filtering problem in Definition 1, and the state estimation problem that stems from it in Definition 2.

Definition 1 (The Filtering Problem)**.** The filtering problem refers to identifying and sequentially updating the probability densities $\pi_j(v_j^\dagger) := \mathbb{P}(v_j^\dagger | Y_j^\dagger)$ in $\mathcal{P}(\mathbb{R}^{d_v})$ for $j \in [J]$. These densities π_j are referred to as the filtering distributions at time j.

Definition 2 (State Estimation)**.** The state estimation problem refers to estimating the state $v_j^\dagger$, based on the given observations $Y_j^\dagger$, and to updating the estimate sequentially over a series of time indices $j \in [J]$.

State estimation can be viewed as a subproblem of the filtering problem. Given the filtering distribution π_j, a natural state estimate is the conditional expectation $\mathbb{E}[v_j^\dagger | Y_j^\dagger]$ as this delivers the minimum mean-squared error estimate of $v_j^\dagger$. In this paper we explicitly focus on *state estimation*: our objective is to recover the hidden state $v_j^\dagger$ at each time step rather than to approximate the full filtering distribution π_j. Accordingly, our loss functions are defined with respect to the true state. We note that, in high-dimensional settings, the conditional mean is generally inaccessible in practice-accurate computation would require particle filters with prohibitively large ensembles-so it is standard to evaluate and tune filters against the true state. As outlined in Section 6, our framework can be extended to the filtering problem by adopting distribution-matching losses to learn the filtering distribution (and hence its mean).

2.2. Filtering: Probability evolution

We consider the filtering problem, which consists of identifying the filtering distribution given by

$$\pi_j(v_j^\dagger) := \mathbb{P}(v_j^\dagger | Y_j^\dagger). \tag{2.3}$$

We define the following two measures

$$\hat{\pi}_{j+1}(v_{j+1}^\dagger) = \mathbb{P}(v_{j+1}^\dagger | Y_j^\dagger), \tag{2.4a}$$

$$\rho_{j+1}(v_{j+1}^\dagger, y_{j+1}^\dagger) = \mathbb{P}(v_{j+1}^\dagger, y_{j+1}^\dagger | Y_j^\dagger). \tag{2.4b}$$

The update $\pi_j \to \pi_{j+1}$ can be written in the following three steps:

Predict (linear): $$\hat{\pi}_{j+1} = \mathsf{P}\pi_j, \tag{2.5a}$$

Extend to data/state space (linear): $$\rho_{j+1} = \mathsf{Q}\hat{\pi}_{j+1}, \tag{2.5b}$$

Analysis (nonlinear): $$\pi_{j+1} = \mathsf{B}(\rho_{j+1}; y_{j+1}^\dagger). \tag{2.5c}$$

P is a linear operator that transforms the filtering distribution at time j to the forecast distribution at time $j+1$, and is given for the stochastic dynamics model in (2.1) by:

$$\mathsf{P}\pi(v) = \frac{1}{\sqrt{(2\pi)^{d_v} \det \Sigma}} \int \exp\left(-\frac{1}{2}\|v - \Psi(u)\|_\Sigma^2\right)\pi(u)\, du. \tag{2.6}$$

Q is also a linear operator, one that multiplies the forecast distribution by the observation likelihood, and is given by:

$$\mathsf{Q}\pi(v, y) = \frac{1}{\sqrt{(2\pi)^{d_y} \det \Gamma}} \exp\left(-\frac{1}{2}\|y - h(v)\|_\Gamma^2\right)\pi(v). \tag{2.7}$$

Then, the nonlinear operator B can be written in the form:

$$\mathsf{B}(\rho_{j+1}; y_{j+1}^\dagger)(v) = \frac{\rho_{j+1}(v, y_{j+1}^\dagger)}{\int_{\mathbb{R}^{d_v}} \rho_{j+1}(u, y_{j+1}^\dagger)\, du}. \tag{2.8}$$

2.3. Filtering: State-space evolution

Directly evolving the filtering distribution π_j is generally intractable since the nonlinear operator B (2.8) cannot be computed explicitly for most practical applications. Alternatively, we consider a random state evolution process governed by its own law, given by

Predict: $$\hat{v}_{j+1} = \Psi(v_j) + \xi_j, \quad \xi_j \sim \mathsf{N}(0, \Sigma), \tag{2.9a}$$

Extend to observation: $$\hat{y}_{j+1} = h(\hat{v}_{j+1}) + \eta_{j+1}, \quad \eta_{j+1} \sim \mathsf{N}(0, \Gamma), \tag{2.9b}$$

Analysis: $$v_{j+1} = \mathfrak{B}(\hat{v}_{j+1}, \hat{y}_{j+1}; y_{j+1}^\dagger, \rho_{j+1}), \tag{2.9c}$$

where ρ_{j+1} is the joint measure defined by (2.4b). From the definitions of P (2.6) and Q (2.7), and from Eqs. (2.9a) and (2.9b), it follows that $(\hat{v}_{j+1}, \hat{y}_{j+1})$ is distributed as ρ_{j+1}, i.e., $\rho_{j+1} = \mathrm{Law}\big((\hat{v}_{j+1}, \hat{y}_{j+1})\big)$. Now, using the theory of transport, we choose $\mathfrak{B}$ so that the pushforward of ρ_{j+1} under this map delivers the desired conditioning on the observed data $y_{j+1}^\dagger$:

$$\mathsf{B}(\rho_{j+1}; y_{j+1}^\dagger) = \mathfrak{B}(\bullet, \bullet, ; y_{j+1}^\dagger, \rho_{j+1})_\sharp(\rho_{j+1}).$$

It follows that if $\pi_j = \mathrm{Law}(v_j)$ then $\pi_{j+1} = \mathrm{Law}(v_{j+1})$.

The preceding construct is mathematically appealing but it leaves open the question of how to identify an appropriate choice of $\mathfrak{B}$. The mean-field ensemble Kalman filter [48] leaves (2.9a) and (2.9b) intact but replaces the mapping $\mathfrak{B}$ in (2.9c) by the affine (in $(\hat{v}_{j+1}, \hat{y}_{j+1})$) approximation

$$v_{j+1} = \mathfrak{B}_{\mathrm{EnKF}}(\hat{v}_{j+1}, \hat{y}_{j+1}; y_{j+1}^\dagger, \rho_{j+1}), \tag{2.10}$$

defined by

$$v_{j+1} = \hat{v}_{j+1} + K_{j+1}\left(y_{j+1}^\dagger - \hat{y}_{j+1}\right), \tag{2.11a}$$

$$K_{j+1} = \hat{C}_{j+1}^{vh}\left(\hat{C}_{j+1}^{hh} + \Gamma\right)^{-1}. \tag{2.11b}$$

Here, K_{j+1} is called the Kalman gain, $\hat{C}^{vh}$ is the covariance between $\hat{v}_{j+1}$ and $h(\hat{v}_{j+1})$, and $\hat{C}_{j+1}^{hh}$ is the covariance for $h(\hat{v}_{j+1})$. Both $\hat{C}^{vh}$ and $\hat{C}_{j+1}^{hh}$ are computed under $\hat{\pi}_{j+1}$:

$$\hat{C}_{j+1}^{vh} = \mathbb{E}^{\hat{v}\sim\hat{\pi}_{j+1}}\left[\left(\hat{v} - \bar{v}_{j+1}\right) \otimes \left(h(\hat{v}) - \bar{h}_{j+1}\right)\right], \tag{2.12a}$$

$$\hat{C}_{j+1}^{hh} = \mathbb{E}^{\hat{v}\sim\hat{\pi}_{j+1}}\left[\left(h(\hat{v}) - \bar{h}_{j+1}\right) \otimes \left(h(\hat{v}) - \bar{h}_{j+1}\right)\right]. \tag{2.12b}$$

Here, $\bar{v}_{j+1}$ and $\bar{h}_{j+1}$ are the mean of states and observations respectively, under $\hat{\pi}_{j+1}$:

$$\bar{v}_{j+1} = \mathbb{E}^{\hat{v}\sim\hat{\pi}_{j+1}}\hat{v}, \quad \bar{h}_{j+1} = \mathbb{E}^{\hat{v}\sim\hat{\pi}_{j+1}}h(\hat{v}). \tag{2.13}$$

We note that in computation of these covariances we have simply used expectations under $\hat{\pi}_{j+1}$, the marginal of ρ_{j+1} on the state. It is possible to use the replacement

$$\hat{C}^{yy}_{j+1} = \hat{C}^{hh}_{j+1} + \Gamma,$$

where $\hat{C}^{yy}_{j+1}$ is the covariance for $\hat{y}_{j+1}$. Hence we write the map $\mathfrak{B}_{\mathrm{EnKF}}$ as dependent on ρ_{j+1}, which covers both cases.

What is sometimes termed the stochastic ensemble Kalman filter is implemented in practice by employing an interacting particle system approximation of the mean-field model. The methodology uses the empirical measure defined by the set of N particles $\{\upsilon_j^{(\ell)}\}_{\ell=1}^N$ to approximate the filtering distribution $\pi_j = \mathrm{Law}(\upsilon_j)$, and hence the empirical measure defined by $\left\{\hat{\upsilon}^{(\ell)}_{j+1}\right\}_{\ell=1}^N$ and $\left\{\left(\hat{\upsilon}^{(\ell)}_{j+1}, \hat{y}^{(\ell)}_{j+1}\right)\right\}_{\ell=1}^N$, to approximate $\hat{\pi}_{j+1}$ and ρ_{j+1}, respectively. We write the resulting approximations as

$$\hat{\pi}^{(N)}_{j+1} := \frac{1}{N}\sum_{n=1}^N \delta_{\hat{\upsilon}^{(n)}_{j+1}}, \quad \rho^{(N)}_{j+1} := \frac{1}{N}\sum_{n=1}^N \delta_{(\hat{\upsilon}^{(n)}_{j+1}, \hat{y}^{(n)}_{j+1})}. \tag{2.14}$$

Making this particle approximation in (2.9), and replacing $\mathfrak{B}$ by $\mathfrak{B}_{\mathrm{EnKF}}$ leads to the interacting particle system

$$\hat{\upsilon}^{(n)}_{j+1} = \Psi(\upsilon^{(n)}_j) + \xi^{(n)}_j, \tag{2.15a}$$

$$\hat{y}^{(n)}_{j+1} = h(\hat{\upsilon}^{(n)}_{j+1}) + \eta^{(n)}_{j+1}, \tag{2.15b}$$

$$\upsilon^{(n)}_{j+1} = \mathfrak{B}_{\mathrm{EnKF}}\left(\hat{\upsilon}^{(n)}_{j+1}, \hat{y}^{(n)}_{j+1}; y^\dagger_{j+1}, \rho^{(N)}_{j+1}\right). \tag{2.15c}$$

This is a particular instance of the ensemble Kalman filter (EnKF), where use of the empirical approximation $\rho^{(N)}_{j+1}$ means that the covariances $\hat{C}^{vh}_{j+1}$ and $\hat{C}^{hh}_{j+1}$ given by (2.12) are now computed by use of empirical approximation $\hat{\pi}^{(N)}_{j+1}$ of $\hat{\pi}_{j+1}$:

$$\hat{C}^{vh}_{j+1} = \mathbb{E}^{\hat{\upsilon}\sim\hat{\pi}^{(N)}_{j+1}}\left[\left(\hat{\upsilon} - \bar{\upsilon}_{j+1}\right) \otimes \left(h(\hat{\upsilon}) - \bar{h}_{j+1}\right)\right] = \frac{1}{N}\sum_{n=1}^N \left(\hat{\upsilon}^{(n)}_{j+1} - \bar{\upsilon}_{j+1}\right) \otimes \left(h(\hat{\upsilon}^{(n)}_{j+1}) - \bar{h}_{j+1}\right), \tag{2.16a}$$

$$\hat{C}^{hh}_{j+1} = \mathbb{E}^{\hat{\upsilon}\sim\hat{\pi}^{(N)}_{j+1}}\left[\left(h(\hat{\upsilon}) - \bar{h}_{j+1}\right) \otimes \left(h(\hat{\upsilon}) - \bar{h}_{j+1}\right)\right] = \frac{1}{N}\sum_{n=1}^N \left(h(\hat{\upsilon}^{(n)}_{j+1}) - \bar{h}_{j+1}\right) \otimes \left(h(\hat{\upsilon}^{(n)}_{j+1}) - \bar{h}_{j+1}\right). \tag{2.16b}$$

Here $\bar{\upsilon}_{j+1}$ and $\bar{h}_{j+1}$, originally defined in (2.13), are also computed by use of an empirical approximation of $\hat{\pi}_{j+1}$:

$$\bar{\upsilon}_{j+1} = \mathbb{E}^{\hat{\upsilon}\sim\hat{\pi}^{(N)}_{j+1}}\hat{\upsilon} = \frac{1}{N}\sum_{n=1}^N \hat{\upsilon}^{(n)}_{j+1}, \qquad \bar{h}_{j+1} = \mathbb{E}^{\hat{\upsilon}\sim\hat{\pi}^{(N)}_{j+1}} h(\hat{\upsilon}) = \frac{1}{N}\sum_{n=1}^N h(\hat{\upsilon}^{(n)}_{j+1}). \tag{2.17a}$$

Remark 3. The ensemble Kalman filter exhibits permutation invariance with respect to the ensemble members. This important property stems from the fact that both the sample means $\bar{\upsilon}_{j+1}$, $\bar{h}_{j+1}$ in (2.17) and the empirical covariance matrices $\hat{C}^{vh}_{j+1}$, $\hat{C}^{hh}_{j+1}$ in (2.16) are calculated as averages over all ensemble members, making them invariant to the ordering of particles in the ensemble. This permutation invariance is a crucial property that motivates the design of our methodology.

2.4. Learning probability measure–dependent analysis maps

Machine learning–based approaches can be applied to approximate the analysis step (2.5c). Indeed, one may seek an operator $\mathsf{B}_{\mathrm{MNM}}$ (*measure neural mappings*, or MNMs, will be introduced in Section 3) acting on probability measures so that

$$\mathsf{B}_{\mathrm{MNM}}(\rho_{j+1}; y^\dagger_{j+1}) \approx \mathsf{B}(\rho_{j+1}; y^\dagger_{j+1}). \tag{2.18}$$

This aim motivates a novel and significant extension of neural operators [49,50]: designing operator architectures that act on the space of probability measures. We refer to these neural operators that act on probability measures as *measure neural mappings* (MNM). A natural way to construct such an approximation is by neural operator $\mathfrak{B}_{\mathrm{MNM}}$, acting on the joint space of state and data and parameterized by the observation and by the law of the state, so that

$$\mathfrak{B}_{\mathrm{MNM}}(\hat{\upsilon}_{j+1}, \hat{y}_{j+1}; y^\dagger_{j+1}, \rho_{j+1}) \approx \mathfrak{B}(\hat{\upsilon}_{j+1}, \hat{y}_{j+1}; y^\dagger_{j+1}, \rho_{j+1}), \tag{2.19a}$$

$$\mathsf{B}_{\mathrm{MNM}}(\rho_{j+1}; y^\dagger_{j+1}) = \mathfrak{B}_{\mathrm{MNM}}(\bullet, \bullet; y^\dagger_{j+1}, \rho_{j+1})_\sharp \rho_{j+1} \approx \mathsf{B}(\rho_{j+1}; y^\dagger_{j+1}). \tag{2.19b}$$

The map $\mathfrak{B}_{\mathrm{MNM}}$ in (2.19a) acts on the joint state–observation space; in contrast the map $\mathsf{B}_{\mathrm{MNM}}$ in (2.19b), defined through pushforward, acts on the space of probability measures on the state space. The quantities after the semi-colon parameterize these maps. To use this neural operator approximation in the state-space evolution, we replace (2.9c) by the approximation

$$\upsilon_{j+1} = \mathfrak{B}_{\mathrm{MNM}}(\hat{\upsilon}_{j+1}, \hat{y}_{j+1}; y^\dagger_{j+1}, \rho_{j+1}). \tag{2.20}$$

2.5. EnKF-inspired probability measure–dependent analysis maps

We now propose a specific novel filtering methodology, which we coin as the *measure neural mapping enhanced ensemble filter* (MNMEF). In this section, we will discuss the mean-field formulation of this approach. We consider a specific form of $\mathfrak{B}_{\text{MNM}}$ inspired by the success of the EnKF and describe it here in the mean-field limit. We generalize (2.11) to take the form

$$v_{j+1} = \hat{v}_{j+1} + (K_\theta)_{j+1}\left(y^\dagger_{j+1} - \hat{y}_{j+1}\right), \tag{2.21a}$$

$$(K_\theta)_{j+1} = K_\theta\left(\rho_{j+1}, y^\dagger_{j+1}\right), \tag{2.21b}$$

where $K_\theta\left(\rho_{j+1}, y^\dagger_{j+1}\right)$ is a parameterized $\mathbb{R}^{d_v \times d_y}$-valued function, and θ denotes the vector of trainable parameters. Our proposed choice for $\mathfrak{B}_{\text{MNM}}$ (2.21a) introduces inductive bias by mimicking the affine EnKF update. The following proposed form for $K_\theta(\bullet,\bullet)$ is identical at each time step; for simplicity we omit the subscript $j+1$ in its definition and, in particular, use $\hat{\pi}$ as the distribution of the predicted state. We assume the following form for K_θ, generalizing the Kalman gain (2.11b):

$$K_\theta = K^{(1)}_\theta\left(K^{(2)}_\theta + \Gamma\right)^{-1}. \tag{2.22}$$

Here $K^{(1)}_\theta$ and $K^{(2)}_\theta$ are modified versions of $\hat{C}^{vh}$ and $\hat{C}^{hh}$ from (2.12). Specifically we have

$$K^{(1)}_\theta = \mathbb{E}^{\hat{v}\sim\hat{\pi}}\left[\left(\hat{v} - \bar{v} + \hat{w}_\theta\right) \otimes \left(h(\hat{v}) - \bar{h} + \hat{z}_\theta\right)\right], \tag{2.23a}$$

$$K^{(2)}_\theta = \mathbb{E}^{\hat{v}\sim\hat{\pi}}\left[\left(h(\hat{v}) - \bar{h} + \hat{z}_\theta\right) \otimes \left(h(\hat{v}) - \bar{h} + \hat{z}_\theta\right)\right], \tag{2.23b}$$

where $\hat{w}_\theta$ and $\hat{z}_\theta$ are trainable correction terms that depend on the state $\hat{v}$, observation $h(\hat{v})$, true observation $y^\dagger$, and the joint distribution ρ_h. We apply a neural operator $\mathfrak{F}(\cdot;\theta) : \mathbb{R}^{d_v} \times \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \times \mathcal{P}(\mathbb{R}^{d_b} \times \mathbb{R}^{d_y}) \to \mathbb{R}^{d_v} \times \mathbb{R}^{d_y}$ to define the correction terms as

$$\mathfrak{F}\left(\hat{v}, h(\hat{v}), y^\dagger, \rho_h; \theta\right) = \left(\hat{w}_\theta, \hat{z}_\theta\right), \tag{2.24}$$

where $\rho_h = \text{Law}\left((\hat{v}, h(\hat{v}))\right)$ is the joint distribution of $(\hat{v}, h(\hat{v}))$. As well as being defined in the mean-field limit, this form of the model can both be learned and deployed through interacting particle system approximations, just as for the original ensemble Kalman filter. These interacting particle systems, as well as details of the architecture employed for $\mathfrak{F}$ in the particle approximation, are defined in Section 4.

Remark 4 (Factoring Our Γ)**.** We note that K_θ (2.21b) depends on the probability measure ρ while $\mathfrak{F}$ (2.24) depends on ρ_h. The relationship between ρ and ρ_h is given by

$$(\hat{v}, \hat{y}) \sim \rho \;\Leftrightarrow\; \begin{cases} (\hat{v},\, h(\hat{v})) \sim \rho_h, \\ \eta \sim \mathcal{N}(0, \Gamma), \quad \eta \perp \hat{v}, \\ \hat{y} = h(\hat{v}) + \eta. \end{cases} \tag{2.25}$$

By factoring out the noise covariance in the term $(K^{(2)}_\theta + \Gamma)^{-1}$ in (2.22), we use the explicit knowledge of Γ in derivation of K_θ. Therefore, $\mathfrak{F}$ uses the input ρ_h which does not depend on Γ.

Remark 5 (Beyond Gaussian Assumptions)**.** The mean-field map $\mathfrak{B}_{\text{MNM}}$ (2.21a) is constrained to be affine in both inputs $\hat{v}$ and $\hat{y}$. Indeed,

$$\mathfrak{B}_{\text{MNM}}(\hat{v}, \hat{y}; y^\dagger, \rho) = \hat{v} + K_\theta\left(y^\dagger - \hat{y}\right).$$

We note that this is the same constraint satisfied by the ensemble Kalman filter; see [48]. However, as shown in [48], the gain K in the ensemble Kalman filter is such that the resulting state estimate has law with first and second-order moments matching a Gaussian approximation of the true filter. Our approach is more general. Indeed, since K_θ is trainable, unlike in the EnKF, the gain is not constrained to satisfy any moment-matching with a Gaussian approximation of the true filter. The ensemble Kalman filter possesses statistical guarantees for problems involving filtering distributions that are close to Gaussian (given, for example, by dynamics and observation operators that are close to linear) [51]. Further work will involve investigating whether this more general methodology possesses theoretical guarantees for a broader class of filtering problems than the EnKF.

The core of our proposed approach is to learn a neural operator $\mathfrak{F}$, as given in (2.24), which takes both probability measures, as well as vectors, as inputs. Section 3 is a self-contained description of a novel methodology for training neural operators that take probability measures as inputs; it leverages the attention mechanism [38]. In Section 4 we will build on this novel neural operator framework to provide a concrete methodology for enhanced data assimilation when actioned using interacting particle approximations of the mean-field limit.

3. Learning maps on the space of probability measures

In this section we introduce a general framework to define neural network architectures acting on the space of probability measures. Let Ω_1, Ω_2 denote two sets, equipped with sigma algebras so that we may assign probabilities, and consider maps of the form $\mathsf{F} : \mathcal{P}(\Omega_1) \times \Theta \to \mathcal{P}(\Omega_2)$; here $\Theta \subseteq \mathbb{R}^p$. Our goal is to define the parametric class of functions so that, by choice of $\theta^* \in \Theta$, we may

approximate a given map $\mathsf{F}^\dagger : \mathcal{P}(\Omega_1) \to \mathcal{P}(\Omega_2)$ by $\mathsf{F}(\cdot;\theta^*) : \mathcal{P}(\Omega_1) \to \mathcal{P}(\Omega_2)$. As the desired size of the approximation error shrinks we will typically require Θ, and in particular the number of parameters p, to grow. Such methodology is now well-established for maps between Banach spaces; see the review [50]. Defining such maps between metric spaces in general, and probability spaces in particular, is an unexplored research question which we address here. We refer to neural networks on the space of probability measures as *measure neural mappings* (MNM).

In what follows, we show that the popular transformer architecture may be formulated as a mean-field mapping on the space of probability measures; this mapping is mean-field because it not only acts on but is also parametrized by probability measures. We call such a neural network architecture the *transformer measure neural mapping* (TMNM). This architecture will serve as a parametric family of operators of the form $\mathsf{F} : \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}(\mathbb{R}^{d_w}) \times \Theta \to \mathcal{P}(\mathbb{R}^{d_v})$ defined via action on $\mu \in \mathcal{P}(\mathbb{R}^{d_u})$ and $\nu \in \mathcal{P}(\mathbb{R}^{d_w})$ as

$$\mathsf{F}(\mu, \nu; \theta) = \mathfrak{F}(\bullet\,; \nu, \theta)_\sharp \mu. \tag{3.1}$$

The transport map in (3.1) is a mapping of the form $\mathfrak{F} : \mathbb{R}^{d_u} \times \mathcal{P}(\mathbb{R}^{d_w}) \times \Theta \to \mathbb{R}^{d_v}$. For ease of exposition we will henceforth omit the θ parameterizations of the operators. In Section 4 we will show how this transformer measure neural mapping, in particular the transport map that defines it, may be used to implement (2.24) in the context of learning an ensemble-based data assimilation algorithm. For this section, however, we work quite generally, without specific reference to data assimilation.

Remark 6 (Training a Transformer MNM)**.** In practice, implementing MNM architectures such as that in (3.1) involves working with samples from the measures $\mu \in \mathcal{P}(\mathbb{R}^{d_u})$ and $\nu \in \mathcal{P}(\mathbb{R}^{d_w})$ rather than the measures themselves. This finite sample approximation introduces the challenge of selecting an appropriate loss function that is readily computed from samples. For the purposes of this article, we will employ a loss function based on matching the mean of the output measure under the map. Future work will optimize based on the use of metrics on probability measures and scoring rules, not just the mean; see Section 4 for more details.

In what follows we assume access to samples $u(i) \sim \mu$ for $i = 1, \dots, N$ and similarly $w(j) \sim \nu$ for $j = 1, \dots, M$. We note the following useful identity, which follows from (3.1) (dropping explicit $\theta-$dependence):

$$\mathsf{F}\Big(\frac{1}{N}\sum_{i=1}^{N} \delta_{u(i)}, \nu\Big) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\mathfrak{F}\left(u(i);\nu\right)}. \tag{3.2}$$

This characterization will be used to describe implementable algorithms.

In view of Remark 6 we proceed by describing the TMNM architecture in detail as a mapping between spaces of probability measures. This mean-field perspective is more general and justifies the deployment of MNM architectures trained with a finite collection of samples to ensembles of different sizes. Nonetheless, in the development that follows, we provide examples illustrating the finite sample empirical setting such as in Remark 6, which will be the focus of later sections.

We proceed in Section 3.1 by describing the TMNM architecture as a composition of specific maps on probability measures, which we call attention blocks and which involve the application of the attention map. In Section 3.2 we define attention as a map on measures and show how, via application of this map to empirical measures, it is possible to recover the formulation of attention on Euclidean spaces from [38].

3.1. Transformer measure neural mapping

The TMNM architecture as an operator of the form $\mathsf{F} : \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}(\mathbb{R}^{d_w}) \to \mathcal{P}(\mathbb{R}^{d_u})$ consists of a composition of $N_e \in \mathbb{N}$ number of operators $\mathsf{S}_w : \mathcal{P}(\mathbb{R}^{d_w}) \to \mathcal{P}(\mathbb{R}^{d_w})$, $N_d \in \mathbb{N}$ number of operators $\mathsf{S}_u : \mathcal{P}(\mathbb{R}^{d_u}) \to \mathcal{P}(\mathbb{R}^{d_u})$, and $\mathsf{C} : \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}(\mathbb{R}^{d_w}) \to \mathcal{P}(\mathbb{R}^{d_u})$, which we call the self-attention and cross-attention blocks, respectively. We define F via action on inputs $\mu \in \mathcal{P}(\mathbb{R}^{d_u})$ and $\nu \in \mathcal{P}(\mathbb{R}^{d_w})$ as

$$\mathsf{F}(\mu, \nu) = \mathsf{S}_{u,N_d+N_e}(\bullet) \circ \cdots \circ \mathsf{S}_{u,1+N_e}(\bullet) \circ \mathsf{C}(\mu, \bullet) \circ \mathsf{S}_{w,N_e}(\bullet) \circ \cdots \circ \mathsf{S}_{w,2}(\bullet) \circ \mathsf{S}_{w,1}(\nu). \tag{3.3}$$

For ease of exposition, for operators S we will henceforth drop the notation $\mathsf{S}_\bullet$, as the dimension of the input state space will be clear from context. We note that the cross-attention block is intimately related to the pooling multihead attention block from the set transformer architecture [52]. In Remark 7 and in Section 4 we make explicit the connection between the TMNM and the set transformer. Both operators S and C will be defined via a pushforward operation under a transport map $\mathfrak{C}$: we define

$$\mathsf{S}(\mu) = \mathfrak{C}(\bullet; \mu)_\sharp \mu, \tag{3.4a}$$

$$\mathsf{C}(\mu, \nu) = \mathfrak{C}(\bullet; \nu)_\sharp \mu. \tag{3.4b}$$

We now outline the construction of the transport map appearing in (3.4a) and (3.4b). Indeed, we may write the map $\mathfrak{C} : \mathbb{R}^{d_u} \times \mathcal{P}(\mathbb{R}^{d_w}) \to \mathbb{R}^{d_u}$ acting on the inputs $u \in \mathbb{R}^{d_u}$ and $\pi \in \mathcal{P}(\mathbb{R}^{d_w})$ as the composition of the maps

$$u \mapsto \mathfrak{F}^{\mathrm{LN}}\big(u + \mathfrak{A}(u; \pi)\big), \tag{3.5a}$$

$$u \mapsto \mathfrak{F}^{\mathrm{LN}}\big(u + \mathfrak{F}^{\mathrm{NN}}(u)\big). \tag{3.5b}$$

We note that each of the maps appearing in the composition in (3.5) may be used to define a pushforward operator on measures.

Remark 7 (Set Transformer Architecture)**.** We note that the composition in (3.3) is a generalization of the set transformer architecture from [52] to the continuum. Indeed, the set transformer acts on a collection of vectors $\{u(1), \dots, u(N)\} \subset \mathbb{R}^{d_u}$ which we may assume to be drawn from an underlying reference measure μ. The set transformer relies on learning a vector $w \in \mathbb{R}^{d_w}$ with which cross-attention is applied to the ensemble. The set transformer architecture may thus be viewed as the map F in (3.3) when acting on the empirical measure $\mu^N = \frac{1}{N}\sum_{i=1}^{N} \delta_{u(i)}$ and a learned dirac δ_w.

We will now turn our focus to defining each of the maps in (3.5). The map $\mathfrak{F}^{\mathrm{LN}} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_u}$ defines layer normalization and is such that

$$\left(\mathfrak{F}^{\mathrm{LN}}(u;\gamma,\beta)\right)_k = \gamma_k \cdot \frac{u_k - m(u)}{\sqrt{\sigma^2(u)+\epsilon}} + \beta_k, \tag{3.6}$$

for $k = 1, \dots, d_u$, any $u \in \mathbb{R}^{d_u}$, where the subscript notation $(\bullet)_k$ is used to denote the k'th entry of the vector. In Eq. (3.6), $\epsilon \in \mathbb{R}^+$ is a fixed parameter, $\gamma_k, \beta_k \in \mathbb{R}$ are learnable parameters and m, σ are defined as

$$m(u) = \frac{1}{d_u}\sum_{k=1}^{d_u} u_k, \qquad \sigma^2(u) = \frac{1}{d_u}\sum_{k=1}^{d_u}\big(u_k - m(u)\big)^2, \tag{3.7}$$

for any $u \in \mathbb{R}^{d_u}$. The operator $\mathfrak{F}^{\mathrm{NN}} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_u}$ is defined such that

$$\mathfrak{F}^{\mathrm{NN}}(u; W_1, W_2, b_1, b_2) = W_2 f\left(W_1 u + b_1\right) + b_2, \tag{3.8}$$

for any $u \in \mathbb{R}^{d_u}$, where $W_1, W_2 \in \mathbb{R}^{d_u \times d_u}$ and $b_1, b_2 \in \mathbb{R}^{d_u}$ are learnable parameters and where f is a nonlinear activation function.

The map $\mathfrak{A} : \mathbb{R}^{d_u} \times \mathcal{P}(\mathbb{R}^{d_w}) \to \mathbb{R}^{d_u}$ is the attention map: Section 3.2 which follows is dedicated to its formulation as a transport defining the attention map on measures A.

3.2. Attention as a map on measures

We present the definition of the attention operator as a map between a product of the spaces of probability measures $\mathcal{P}(\mathbb{R}^{d_u})$ and $\mathcal{P}(\mathbb{R}^{d_w})$ into another space of probability measures $\mathcal{P}(\mathbb{R}^{d_v})$. [1] If considering the map as acting on measures defined on the unbounded domain $\mathbb{R}^{d_w}$ we must make assumptions on the input measures themselves, i.e., tail decay assumptions, so that the attention operator is well-defined. Hence, for ease of exposition we first present attention as an operator $\mathsf{A} : \mathcal{P}(\Omega_u) \times \mathcal{P}(\Omega_w) \to \mathcal{P}(\mathbb{R}^{d_v})$, where Ω_u, Ω_w are bounded open subsets of $\mathbb{R}^{d_u}, \mathbb{R}^{d_w}$ respectively; we will then extend the definition to the general case $\mathsf{A} : \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}_\delta(\mathbb{R}^{d_w}) \to \mathcal{P}(\mathbb{R}^{d_v})$, where the subset of measures $\mathcal{P}_\delta(\mathbb{R}^{d_w}) \subset \mathcal{P}(\mathbb{R}^{d_w})$ consisting of all probability measures whose tails decay strictly faster than exponential, will be appropriately defined.

Definition 3 (Attention on Measures: Bounded State Space)**.** We define the operator $\mathsf{A} : \mathcal{P}(\Omega_u) \times \mathcal{P}(\Omega_w) \to \mathcal{P}(\mathbb{R}^{d_v})$ via its action on inputs $(\mu, \nu) \in \mathcal{P}(\Omega_u) \times \mathcal{P}(\Omega_w)$ as a pushforward of the first input measure $\mu \in \mathcal{P}(\Omega_u)$ by a transport map $\mathfrak{A} : \Omega_u \times \mathcal{P}(\Omega_w) \to \mathbb{R}^{d_v}$ which is parametrized by the second input measure $\nu \in \mathcal{P}(\Omega_w)$, hence

$$\mathsf{A}(\mu, \nu) = \mathfrak{A}(\cdot; \nu)_\sharp \mu. \tag{3.9}$$

The transport map $\mathfrak{A} : \Omega_u \times \mathcal{P}(\Omega_w) \to \mathbb{R}^{d_v}$ takes as input a vector $u \in \Omega_u$ and a probability measure $\nu \in \mathcal{P}(\Omega_w)$ and computes an expectation under a rescaling of the measure ν, which is parametrized by u. Namely for $(u, \nu) \in \Omega_u \times \mathcal{P}(\Omega_w)$ it holds that

$$\mathfrak{A}(u; \nu) = \mathbb{E}_{w \sim p(\cdot; u, \nu)} V w, \tag{3.10}$$

where

$$p(dw; u, \nu) = \frac{\exp\big(\langle Qu, Kw\rangle\big)\nu(dw)}{\int_{\Omega_w} \exp\big(\langle Qu, Kz\rangle\big)\nu(dz)}. \tag{3.11}$$

Dimensions of the learnable matrices Q, K, V are determined from the above, provided Q, K have the same output dimension, which is a free parameter to be specified.

Remark 8 ($p(\bullet; u)$ is a probability measure)**.** Clearly, the normalization constant in (3.11) is finite, as the integral is taken over Ω_w, a bounded domain. Furthermore p integrates to 1 over Ω_w, hence it defines a probability measure.

In view of Remark 8, in order to extend the definition of the attention operator expressed in (3.9) to a map on probability measures defined on an unbounded domain, extra care is required to ensure the normalization constant in (3.11) is finite. Indeed, we must assume that the measure $\nu \in \mathcal{P}(\mathbb{R}^{d_w})$ has sufficiently fast tail decay in order to compensate for the exponential growth in the integrand. To this end, we introduce the following subset of $\mathcal{P}(\mathbb{R}^{d_w})$, on which the attention operator will be shown to be well-defined. Intuitively, the condition that defines the subset is satisfied by all measures whose tails decay faster than exponentially.

Definition 4 (Set of Measures with Exponential Tail Decay)**.** For $\delta > 0$ we define $\mathcal{P}_\delta(\mathbb{R}^{d_u}) \subset \mathcal{P}(\mathbb{R}^{d_u})$ as a set of measures satisfying the following exponential tail decay condition:

$$\mathcal{P}_\delta(\mathbb{R}^{d_u}) = \left\{\mu \in \mathcal{P}(\mathbb{R}^{d_u}) : \ \mu\{\omega \in \Omega : |X(\omega)| \geq t\} \leq 2\exp\left(-t^{1+\delta}\right)\right\}.$$

Given Definition 4, we may now define the attention operator as a map on probability measures defined over unbounded state spaces.

[1] In this self-contained subsection we use the notation d_v to denote the dimension associated with the output space of probability measures. This is not to be confused with the dimension of the state of the dynamical system (2.1) that is subject of the majority of the remainder of the paper.

Definition 5 (Attention on Measures: Unbounded State Space)**.** We define the operator $\mathsf{A} : \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}_\delta(\mathbb{R}^{d_w}) \to \mathcal{P}(\mathbb{R}^{d_v})$ via its action on inputs $(\mu, \nu) \in \mathcal{P}(\mathbb{R}^{d_u}) \times \mathcal{P}_\delta(\mathbb{R}^{d_w})$ as a pushforward of the first input measure $\mu \in \mathcal{P}(\mathbb{R}^{d_u})$ by a transport map $\mathfrak{A} : \mathbb{R}^{d_u} \times \mathcal{P}_\delta(\mathbb{R}^{d_w}) \to \mathbb{R}^{d_v}$ which is parametrized by the second input measure $\nu \in \mathcal{P}_\delta(\mathbb{R}^{d_w})$, hence

$$\mathsf{A}(\mu, \nu) = \mathfrak{A}(\cdot; \nu)_\sharp \mu. \tag{3.12}$$

The transport map $\mathfrak{A} : \mathbb{R}^{d_u} \times \mathcal{P}_\delta(\mathbb{R}^{d_w}) \to \mathbb{R}^{d_v}$ takes as input a vector $u \in \mathbb{R}^{d_u}$ and a probability measure $\nu \in \mathcal{P}_\delta(\mathbb{R}^{d_w})$ and involves computation of an expectation under a rescaling of the measure ν which is parametrized by u. Namely for $(u, \nu) \in \mathbb{R}^{d_u} \times \mathcal{P}_\delta(\mathbb{R}^{d_w})$ it holds that

$$\mathfrak{A}(u; \nu) = \mathbb{E}_{w \sim p(\cdot; u, \nu)} V w, \tag{3.13}$$

where

$$p(dw; u, \nu) = \frac{\exp\big(\langle Qu, Kw\rangle\big)\nu(dw)}{\int_{\mathbb{R}^{d_w}} \exp\big(\langle Qu, Kz\rangle\big)\nu(dz)}. \tag{3.14}$$

Definition 4 allows to state and prove the following lemma regarding the finiteness of the normalization constant appearing in (3.14).

Lemma 1. *Let Q, K, V be fixed matrices of appropriate dimensions. The measure $p(du; s)$ is a well-defined probability measure for $\nu \in \mathcal{P}_\delta(\mathbb{R}^{d_u})$.*

Proof. It suffices to show $\int_{\mathbb{R}^{d_w}} \exp\big(\langle Qu, Kz\rangle\big)\nu(dz) < \infty$. Using the Cauchy–Schwarz and Young's inequalities we obtain that,

$$\exp\big(\langle Qu, Kz\rangle\big) \le \exp\big(\|Q\|\|K\||u||z|\big) \le \exp\left(\frac{(\|Q\|\|K\||u|)^{\frac{1+\delta}{\delta}}}{(1+\delta)/\delta}\right)\exp\left(\frac{|z|^{1+\delta}}{(1+\delta)}\right). \tag{3.15}$$

We note that

$$\begin{aligned}
\int_{\mathbb{R}^{d_w}} \exp\big(\langle Qu, Kz\rangle\big)\nu(dz) &\le C(u)\int_{\mathbb{R}^{d_w}} \exp\left(\frac{|z|^{1+\delta}}{(1+\delta)}\right)\nu(dz) \\
&= C(u)\int_0^\infty \nu\left(\exp\left(\frac{|z|^{1+\delta}}{(1+\delta)}\right) > t\right)dt \\
&= C(u)\int_0^1 \nu\left(\exp\left(\frac{|z|^{1+\delta}}{(1+\delta)}\right) > t\right)dt + C(u)\int_1^\infty \nu\left(\exp\left(\frac{|z|^{1+\delta}}{(1+\delta)}\right) > t\right)dt \\
&\le C(u) + C(u)\int_1^\infty \nu(|z| > \tau)\exp\left(\frac{\tau^{1+\delta}}{(1+\delta)}\right)\tau^\delta d\tau \\
&\le C(u) + C(u)\int_1^\infty \exp\left(\frac{-\delta}{(1+\delta)}\cdot\tau^{1+\delta}\right)\tau^\delta d\tau < \infty,
\end{aligned}$$

where $C(u)$ is a constant depending on u that changes from line to line, and where in the fourth line we used that the first integral is less than or equal to 1 and applied the substitution $t = \exp\big(\tau^{1+\delta}/(1+\delta)\big)$ to the second integral; furthermore, we used $\nu \in \mathcal{P}_\delta(\mathbb{R}^{d_w})$ for the final inequality. □

For implementable algorithms it is useful to consider the finite sample case, where we assume we have access to empirical approximations of the measures $\mu \in \mathcal{P}(\mathbb{R}^{d_u})$ and $\nu \in \mathcal{P}_\delta(\mathbb{R}^{d_w})$ via a collection of samples, or ensembles, $\{u(1), \dots, u(N)\} \subset \mathbb{R}^{d_u}$ and $\{w(1), \dots, w(M)\} \subset \mathbb{R}^{d_w}$. The following proposition provides a description of the attention operator when applied to empirical measures. The resulting expressions will be useful more generally to formulate the trainable algorithms we will use for data assimilation.

Proposition 1 (Attention on Empirical Measures)**.** *Assume that, for $\mu \in \mathcal{P}(\mathbb{R}^{d_u}), \nu \in \mathcal{P}_\delta(\mathbb{R}^{d_w})$, samples $u(1), \dots, u(N)$ are drawn i.i.d. from μ and samples $w(1), \dots, w(M) \sim \nu$ are drawn i.i.d. from ν. Then for $\mu^N = \frac{1}{N}\sum_{j=1}^N \delta_{u(j)}$ and $\nu^M = \frac{1}{M}\sum_{j=1}^M \delta_{w(j)}$ it holds that*

$$\mathsf{A}(\mu^N, \nu^M) = \frac{1}{N}\sum_{j=1}^N \delta_{\mathfrak{A}\left(u(j); \nu^M\right)},$$

where

$$\mathfrak{A}\big(u(j); \nu^M\big) = \frac{\sum_{k=1}^M \exp\big(\langle Qu(j), Kw(k)\rangle\big)Vw(k)}{\sum_{\ell=1}^M \exp\big(\langle Qu(j), Kw(\ell)\rangle\big)}. \tag{3.17}$$

Proof. This follows by application of identity (3.2) to Definition 5. □

We note that the expression (3.17) corresponds to the definition of attention on discrete sequences as highlighted in [38].

4. Particle approximation of learned mean-field filters

In this section, we detail our proposed machine-learning-based DA method, the *measure neural mapping enhanced ensemble filter (MNMEF)*, for the state estimation problem. In particular we build on the mean-field formulation introduced in Section 2, and in Section 2.5 in particular, by introducing an interacting particle system approximation. The resulting machine-learned ensemble approach is based on updating $\{v_j^{(n)}\}_{n=1}^N$ to $\{v_{j+1}^{(n)}\}_{n=1}^N$ through the following steps:

Predict: $$\hat{v}_{j+1}^{(n)} = \Psi(v_j^{(n)}) + \xi_j^{(n)}, \tag{4.1a}$$

Extend to observation space: $$\hat{y}_{j+1}^{(n)} = h(\hat{v}_{j+1}^{(n)}) + \eta_{j+1}^{(n)}, \tag{4.1b}$$

Analysis: $$v_{j+1}^{(n)} = \hat{v}_{j+1}^{(n)} + (K_\theta)_{j+1}\left(y_{j+1}^\dagger - \hat{y}_{j+1}^{(n)}\right). \tag{4.1c}$$

After the prediction steps (4.1a) and (4.1b), we obtain a set of particles $\{(\hat{v}_{j+1}^{(n)}, h(\hat{v}_{j+1}^{(n)}))\}_{n=1}^N$. The computation of K_θ is elaborated through Eqs. (2.22), (2.23), and (2.24) in a mean-field perspective, now extended to also include this particle setting. As in Section 2, we drop time index j in what follows and recall that $\rho_h = \mathrm{Law}\big((\hat{v}, h(\hat{v}))\big)$ is the joint distribution of $(\hat{v}, h(\hat{v}))$ in the mean-field limit. In the particle case, under their implied empirical distribution, we have

$$\rho_h^{(N)} := \frac{1}{N}\sum_{n=1}^N \delta_{\left(\hat{v}_{j+1}^{(n)}, h(\hat{v}_{j+1}^{(n)})\right)} \tag{4.2}$$

and use this as an input measure to K_θ. The core idea is to train a parameterized neural operator $\mathfrak{F}\big(\hat{v}, h(\hat{v}), y^\dagger, \rho_h; \theta\big)$ that outputs the correction terms used to compute K_θ. This can be done using the transformer measure neural mapping (TMNM) architecture F (3.3) introduced in Section 3. When applied to empirical measures such as (4.2), this TMNM scheme is equivalent to the set transformer architecture [52] acting on sequences, as outlined in Remark 7. In this section, which is concerned with implementation, we will work with the formulation of the scheme on sequences; we will also show that the resulting scheme is invariant to permutation of the elements of the sequence. To distinguish the resulting operator on sequences to the one acting on measures from Section 3, we will employ the notation F^{ST} opposed to F; see (4.18) for the definition of F^{ST}. Due to the permutation invariance property of the set transformer, we can treat the input as an unordered set: we define the corresponding operator acting on finite sets using the notation F^{ST} ; see (4.25).

The parameters θ learned in K_θ implicitly depend on N, the number of ensemble members used in training. However, the relation between the set transformer and its mean-field limit from Section 3 allows us to deploy the model parameters θ, trained using an ensemble size N, to an ensemble of a different sizes N'. This is because the N and N' systems may both be thought of as approximating the same mean-field limit. Similarly to the standard use of the EnKF, our learning framework also incorporates techniques such as inflation and localization to enhance the performance of the filter [6] as ensemble sizes vary. We view this as a fine-tuning: fixing the majority of the parameters in θ as ensemble sizes vary, but allowing a small subset of parameters-including those related to inflation and localization-to vary with ensemble size.

We present the complete implementation details of our method in the following subsections. In Section 4.1, we review the set transformer architecture. In Section 4.2, we present our architecture that learns a parameterized gain matrix, generalizing the EnKF formulation, and enhanced by use of adaptively learned inflation and localization parameters. Section 4.3 explains the loss function and training process. In Section 4.4, we describe how to efficiently fine-tune our method to allow for ensemble size dependent inflation and localization. In A we provide a comprehensive summary of our learning framework with detailed pseudo-code algorithms.

Note that all aspects are based on a discrete setting, where our approach evolves an ensemble of particles, as in the EnKF. Consequently, all measure-related operations are transformed into computations based on the empirical measure, or equivalently, the ensemble, viewed as a set of points in state space. When the time-step subscript is omitted, this indicates that the analysis applies to any time step j.

4.1. The set transformer

In this section, we review the set transformer architecture [52] used to process ensembles $\{(\hat{v}^{(n)})\}_{n=1}^N$ or $\{(h(\hat{v}^{(n)}))\}_{n=1}^N$ into fixed-dimensional feature vectors. While Section 3 introduced the set transformer from a measure-theoretic perspective, we now consider our input as a sequence that, due to the architecture's design, can be treated either as a set or an empirical measure.

For two sequences $u \in \mathcal{U}([N]; \mathbb{R}^{d_u})$ and $w \in \mathcal{U}([M]; \mathbb{R}^{d_w})$, we provide the definition for the sequence-based attention mechanism, which is given by

$$\mathfrak{A}\big(u(j); w\big) = \frac{\sum_{k=1}^M \exp\big(\langle Qu(j), Kw(k)\rangle\big) Vw(k)}{\sum_{\ell=1}^M \exp\big(\langle Qu(j), Kw(\ell)\rangle\big)}. \tag{4.3}$$

We recall that this is equivalent to the formulation (3.17) in Proposition 1 obtained by applying the measure theoretic definition of attention to empirical measures. Defining the set of sequences of finite length in $\mathbb{R}^d$ by,

$$\mathcal{U}_F(\mathbb{R}^d) = \cup_{N=1}^\infty \mathcal{U}([N]; \mathbb{R}^d), \tag{4.4}$$

we can define an attention operator A^{ST} (where the superscript ST indicates it is used in the set transformer architecture) as a sequence-to-sequence operator, analogous to the attention A acting on measures (3.9): [2]

$$\mathsf{A}^{\mathrm{ST}} : \mathcal{U}_F(\mathbb{R}^{d_u}) \times \mathcal{U}_F(\mathbb{R}^{d_w}) \to \mathcal{U}_F(\mathbb{R}^{d_v}). \tag{4.5}$$

In view of (4.3), the output sequence length is the same as the length of the first input. Specifically, for $u \in \mathcal{U}([N];\mathbb{R}^{d_u})$ and $w \in \mathcal{U}([M];\mathbb{R}^{d_w})$,

$$\mathsf{A}^{\mathrm{ST}}(u,w)(j) = \mathfrak{A}\big(u(j);w\big), \quad \forall j \in [N]. \tag{4.6}$$

The learnable parameters in A^{ST} are denoted by

$$\theta(\mathsf{A}^{\mathrm{ST}}) = \{Q, K, V\} \tag{4.7}$$

In the context of practical implementations, A^{ST} is commonly instantiated as a multihead attention mechanism, which allows the model to jointly attend to information from different representation subspaces [38]. The detailed formulation of multihead attention is presented in B.

The set transformer architecture is composed of several key building blocks [52]. The fundamental computation unit is the Cross-Attention Block (CAB) given in the following definition.

Definition 6 (CAB for Sequences)**.** The Cross-Attention Block (CAB) C^{ST} is an operator that maps two sequences into one sequence:

$$\mathsf{C}^{\mathrm{ST}} : \quad \mathcal{U}_F(\mathbb{R}^{d_u}) \times \mathcal{U}_F(\mathbb{R}^{d_w}) \to \mathcal{U}_F(\mathbb{R}^{d_u}). \tag{4.8}$$

For $u \in \mathcal{U}([N];\mathbb{R}^{d_u})$ and $w \in \mathcal{U}([M];\mathbb{R}^{d_w})$, C^{ST} is defined as

$$\mathsf{C}^{\mathrm{ST}}(u,w) = \mathsf{F}_1^{\mathrm{LN}}(u_1 + \mathsf{F}^{\mathrm{NN}}(u_1)) \in \mathcal{U}([N];\mathbb{R}^{d_u}), \tag{4.9a}$$

$$u_1 = \mathsf{F}_2^{\mathrm{LN}}\big(u + \mathsf{A}^{\mathrm{ST}}(u,w)\big) \in \mathcal{U}([N];\mathbb{R}^{d_u}). \tag{4.9b}$$

Here A^{ST} is the sequence-to-sequence attention (4.5); F^{LN} denotes application of the layer normalization operator $\mathfrak{F}^{\mathrm{LN}}$ (3.6), elementwise in the sequence; F^{NN} denotes application of the feedforward operator $\mathfrak{F}^{\mathrm{NN}}$ (3.8), also elementwise in the sequence. The $+$ operator acting on sequences is interpreted as adding elements with the same index, i.e.,

$$(u+v)(j) = u(j) + v(j). \tag{4.10}$$

The trainable parameters in C^{ST} include the parameters from $\mathsf{F}_1^{\mathrm{LN}}$, $\mathsf{F}_2^{\mathrm{LN}}$, F^{NN}, and A^{ST}, i.e.,

$$\theta(\mathsf{C}^{\mathrm{ST}}) = \theta(\mathsf{F}_1^{\mathrm{LN}}) \cup \theta(\mathsf{F}_2^{\mathrm{LN}}) \cup \theta(\mathsf{F}^{\mathrm{NN}}) \cup \theta(\mathsf{A}^{\mathrm{ST}}), \tag{4.11}$$

where the layer normalization layers $\mathsf{F}_1^{\mathrm{LN}}, \mathsf{F}_2^{\mathrm{LN}}$ have trainable parameters according to (3.6)(3.7), the feedforward layer F^{NN} has trainable parameters according to (3.8), and the attention layer A^{ST} has trainable parameters according to (4.7). We note that practically the cross-attention block on sequences is the multi-head attention block from [38,52] and may be viewed as the sequence-to-sequence analog of the cross-attention block C on measures from (3.4b).

The set transformer architecture is composed of several self-attention blocks (SAB) (Definition 7) and a pooling by multi-head attention block (PMA) (Definition 8), which are defined in the following using CAB.

Definition 7 (SAB)**.** The self-attention block (SAB) S^{ST}, also called the set attention block [52], maps a sequence to another sequence with the same length and dimension, so that

$$\mathsf{S}^{\mathrm{ST}} : \mathcal{U}_F(\mathbb{R}^{d_u}) \to \mathcal{U}_F(\mathbb{R}^{d_u}). \tag{4.12}$$

The application of S^{ST} is calculated by setting both inputs of C^{ST} to be the same, i.e.,

$$\mathsf{S}^{\mathrm{ST}}(u) = \mathsf{C}^{\mathrm{ST}}(u,u). \tag{4.13}$$

Here, the trainable parameters satisfy

$$\theta(\mathsf{S}^{\mathrm{ST}}) = \theta(\mathsf{C}^{\mathrm{ST}}). \tag{4.14}$$

See (3.4a) for the measure theoretic analog S.

Definition 8 (PMA)**.** The pooling by multi-head attention block (PMA) $\mathsf{C}^{\mathrm{ST}}(s,\bullet)$ is defined by replacing the first input in the CAB $\mathsf{C}^{\mathrm{ST}}(\bullet,\bullet)$ by a trainable parameter s, called the seed. The seed $s \in \mathcal{U}([N_s],\mathbb{R}^{d_s})$ is a sequence, where N_s and d_s are fixed hyperparameters. Therefore $\mathsf{C}^{\mathrm{ST}}(s,\bullet)$ maps a sequence of any finite length to a sequence with fixed length,

$$\mathsf{C}^{\mathrm{ST}}(s,\bullet) : \mathcal{U}_F(\mathbb{R}^{d_u}) \to \mathcal{U}([N_s];\mathbb{R}^{d_s}). \tag{4.15}$$

The trainable parameters in PMA are the parameters in the CAB and the seed s,

$$\theta\big(\mathsf{C}^{\mathrm{ST}}(s,\bullet)\big) = \theta(\mathsf{C}^{\mathrm{ST}}) \cup \{s\}. \tag{4.16}$$

See Remark 7 for the measure theoretic analog $\mathsf{C}(s,\bullet)$.

[2] As in Section 3, here d_v does not necessarily refer to the state space dimension in the dynamical system; the exposition in this subsection is quite general, defining the set transformer and linking it to the MNM in Section 3.

The set transformer maps a sequence of any finite length to a fixed-dimensional feature vector:

$$\mathsf{F}^{\mathrm{ST}} : \mathcal{U}_F(\mathbb{R}^{d_u}) \to \mathbb{R}^{d_{\mathrm{ST}}}. \tag{4.17}$$

To fully specify the detailed architecture of F^{ST}, we will utilize two multilayer perceptrons (3.8) $\mathsf{F}^{\mathrm{NN}}_{\mathrm{in}} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_{\mathrm{latent}}}$, and $\mathsf{F}^{\mathrm{NN}}_{\mathrm{out}} : \mathbb{R}^{N_s \times d_s} \to \mathbb{R}^{d_{\mathrm{ST}}}$ (that act elementwise on sequences), and a parameter-free concatenation layer $\mathsf{F}^{\mathrm{Cat}} : \mathcal{U}([N_s], \mathbb{R}^{d_s}) \to \mathbb{R}^{N_s \times d_s}$ that concatenates all elements in a sequence into a vector.

For a sequence $u \in \mathcal{U}([N]; \mathbb{R}^{d_u})$, the set transformer is hence given by:

$$\mathsf{F}^{\mathrm{ST}}(u) = \mathsf{F}^{\mathrm{Dec}}(\bullet) \circ \mathsf{C}^{\mathrm{ST}}(s, \bullet) \circ \mathsf{F}^{\mathrm{Enc}}(u), \tag{4.18a}$$

$$\mathsf{F}^{\mathrm{Enc}}(u) = \left(\mathsf{S}^{\mathrm{ST}}_{N_{\mathrm{e}}}(\bullet) \circ \cdots \circ \mathsf{S}^{\mathrm{ST}}_{1}(\bullet)\right) \circ \mathsf{F}^{\mathrm{NN}}_{\mathrm{in}}(u), \tag{4.18b}$$

$$\mathsf{F}^{\mathrm{Dec}}(u) = \mathsf{F}^{\mathrm{NN}}_{\mathrm{out}}(\bullet) \circ \mathsf{F}^{\mathrm{Cat}}(\bullet) \circ \left(\mathsf{S}^{\mathrm{ST}}_{N_{\mathrm{d}}+N_{\mathrm{e}}}(\bullet) \circ \cdots \circ \mathsf{S}^{\mathrm{ST}}_{1+N_{\mathrm{e}}}(\bullet)\right)(u). \tag{4.18c}$$

The subscripts $\ell \in [N_{\mathrm{e}} + N_{\mathrm{d}}]$ emphasize that the trainable parameters of these SABs are different. Integer N_{e} is the number of SABs, $\mathsf{S}^{\mathrm{ST}}_{\ell}, \ell = 1, 2, \ldots, N_{\mathrm{e}}$, in the encoder $\mathsf{F}^{\mathrm{Enc}}$, while N_{d} is the number of SABs, $\mathsf{S}^{\mathrm{ST}}_{\ell}, \ell = 1 + N_{\mathrm{e}}, 2 + N_{\mathrm{e}}, \ldots, N_{\mathrm{d}} + N_{\mathrm{e}}$, in the decoder $\mathsf{F}^{\mathrm{Dec}}$. Specifically, we have

$$\mathsf{S}^{\mathrm{ST}}_{\ell} : \mathcal{U}_F(\mathbb{R}^{d_{\mathrm{latent}}}) \to \mathcal{U}_F(\mathbb{R}^{d_{\mathrm{latent}}}), \qquad \ell = 1, 2, \ldots, N_{\mathrm{e}}, \tag{4.19a}$$

$$\mathsf{S}^{\mathrm{ST}}_{\ell} : \mathcal{U}_F(\mathbb{R}^{d_s}) \to \mathcal{U}_F(\mathbb{R}^{d_s}), \qquad \ell = 1 + N_{\mathrm{e}}, 2 + N_{\mathrm{e}}, \ldots, N_{\mathrm{d}} + N_{\mathrm{e}}. \tag{4.19b}$$

Further practical details about our set transformer architecture, including the dimensions of latent features and the choices of layers, are provided in C.1.

The trainable parameters in F^{ST} are given by:

$$\begin{aligned} \theta(\mathsf{F}^{\mathrm{ST}}) &= \theta\left(\mathsf{F}^{\mathrm{Dec}}\right) \cup \theta\left(\mathsf{C}^{\mathrm{ST}}(s, \bullet)\right) \cup \theta\left(\mathsf{F}^{\mathrm{Enc}}\right) \\ &= \theta\left(\mathsf{F}^{\mathrm{NN}}_{\mathrm{in}}\right) \cup \theta\left(\mathsf{F}^{\mathrm{NN}}_{\mathrm{out}}\right) \cup \left(\cup_{\ell=1}^{N_{\mathrm{e}}+N_{\mathrm{d}}} \theta\left(\mathsf{S}^{\mathrm{ST}}_{\ell}\right)\right) \cup \theta\left(\mathsf{C}^{\mathrm{ST}}(s, \bullet)\right). \end{aligned} \tag{4.20}$$

The set transformer architecture exhibits two key properties when processing an input sequence, namely, the permutation invariance (Proposition 2) and adaptation to variable input lengths (Proposition 3). These two propositions are proved in [52] under a slightly different setting: in the original work a length-N sequence in $\mathbb{R}^d$ is represented as an $N \times d$ matrix, whereas here it is a mapping $u \in \mathcal{U}([N], \mathbb{R}^d)$.

Proposition 2 (Permutation Invariance). *A permutation σ is a bijective function from $[N]$ to $[N]$. We extend the definition of σ to sequences in $u \in \mathcal{U}([N], \mathbb{R}^{d_u})$ so that*

$$\sigma(u)(n) = u(\sigma(n)), \quad n \in [N]. \tag{4.21}$$

Then for any permutation $\sigma : [N] \mapsto [N]$ and any input sequence $u \in \mathcal{U}([N], \mathbb{R}^{d_u})$, we have

$$\mathsf{F}^{ST}(u) = \mathsf{F}^{ST}(\sigma(u)). \tag{4.22}$$

Proposition 3 (Adaptation to Variable Input Length). *A set transformer architecture with fixed parameters takes input sequences with different lengths while the output dimension is fixed:*

$$\mathsf{F}^{ST}(u) \in \mathbb{R}^{d_{ST}}, \quad \forall u \in \mathcal{U}([N], \mathbb{R}^{d_u}), \quad \forall N \in \mathbb{N}. \tag{4.23}$$

Based on these two properties, we can bridge the sequence input and the probability measure input for the set transformer architecture. The permutation invariance property (Proposition 2) enables us to abstract away from the sequential nature of the input and treat it as an unordered set. More fundamentally, this allows us to interpret the input as an empirical distribution, where each element in the sequence represents a sample from some underlying distribution. Furthermore, Proposition 3 demonstrates that we can accommodate varying input lengths without architectural modifications, effectively allowing us to adjust the number of samples in the empirical distribution. Together, these properties (Propositions 2 and 3) provide theoretical foundation for analysis of the main components of the set transformer in the context of probability measures (Section 3).

Remark 9 (Common Notation for Operations on Measures and on Sequences). In this section, much of the notation coincides with that in Section 3. This is because the underlying definitions are almost identical, with the key distinction being that in Section 3 the operators act on probability measures, whereas in this section they act on sequences. The preceding developments in this subsection explain the connection. For example, in Section 3, S and C denote the measure-based self-attention (3.4a) and cross-attention (3.4b) blocks, while in this section, S^{ST} and C^{ST} represent the sequence-based self-attention (4.13) (Definition 7) and cross-attention (4.9) (Definition 6) blocks, respectively. In addition, F refers to the transformer measure neural mapping (3.3) in Section 3 while F^{ST} refers to the set transformer (4.18) acting on sequences.

4.2. Our architecture

In this section we provide a detailed description the architecture in our proposed learnable DA method, MNMEF. The core of our approach is to learn a parameterized gain matrix (Section 4.2.1), generalizing how the classical EnKF utilizes a Gaussian-inspired gain. Additionally, to enhance performance, we incorporate inflation and localization techniques (Sections 4.2.2 and 4.2.3), akin to their role in EnKF.

In our architecture, we use a set transformer (Section 4.1) to process the ensemble $\{(\hat{v}^{(n)}, h(\hat{v}^{(n)}))\}_{n=1}^N$ into a fixed-dimensional feature vector. We define the set of finite subsets in $\mathbb{R}^d$ by,

$$\mathcal{F}(\mathbb{R}^d) = \{A \subseteq \mathbb{R}^d \mid A \text{ contains a finite set of elements}\}. \tag{4.24}$$

Because of the permutation invariance property (Proposition 2), we may reinterpret the set transformer F^{ST} as a map F^{ST} that acts on sets rather than sequences: $F^{\mathrm{ST}} : \mathcal{F}(\mathbb{R}^{d_v} \times \mathbb{R}^{d_y}) \to \mathbb{R}^{d_{\mathrm{ST}}}$, given by

$$F^{\mathrm{ST}}\left(\{(\hat{v}^{(n)}, h(\hat{v}^{(n)}))\}_{n=1}^N; \theta_{\mathrm{ST}}\right) = f_v \in \mathbb{R}^{d_{\mathrm{ST}}} \tag{4.25}$$

The feature vector f_v can be viewed as a representation of the empirical distribution $\rho_h^{(N)}$ given in (4.2) and similarly as an approximate summary of ρ_h. This representation is subsequently utilized to learn the gain matrix, inflation, and the localization.

Remark 10 (Choice of d_{ST})**.** By Proposition 3, F^{ST} outputs a fixed dimension d_{ST} regardless of ensemble size N. Therefore, we do not need to adjust d_{ST} based on N. Larger d_{ST} values offer more expressiveness, but increase computational cost. Our tests with the Lorenz '96 model (Section 5.2) using $d_{\mathrm{ST}} = 32, 64, 128$ showed only marginal improvements with increasing d_{ST}, thus we selected $d_{\mathrm{ST}} = 64$ for all experiments.

4.2.1. *Learning the gain matrix*

From the analysis step (4.1c) and the proposed form (2.22), we have the following form for the learnable analysis step:

$$v^{(n)} = \hat{v}^{(n)} + K_\theta\left(y^\dagger - \hat{y}^{(n)}\right), \tag{4.26a}$$

$$K_\theta = K_\theta^{(1)}\left(K_\theta^{(2)} + \Gamma\right)^{-1}. \tag{4.26b}$$

Recall that, in the mean-field limit, we impose an inductive bias on our architecture by defining $K_\theta^{(1)}$ and $K_\theta^{(2)}$ in the form (2.23), allowing for a learned correction through (2.24). Recall that $\hat{m}$ and $\hat{h}$ are the mean of $\{\hat{v}^{(n)}\}_{n=1}^N$ and $\{h(\hat{v}^{(n)})\}_{n=1}^N$, respectively. The empirical version of (2.23), used in the practical implementation, takes the form

$$K_\theta^{(1)} = \frac{1}{N}\sum_{n=1}^N \left(\hat{v}^{(n)} - \hat{m} + \iota\hat{w}_\theta^{(n)}\right) \otimes \left(h(\hat{v}^{(n)}) - \hat{h} + \hat{z}_\theta^{(n)}\right), \tag{4.27a}$$

$$K_\theta^{(2)} = \frac{1}{N}\sum_{n=1}^N \left(h(\hat{v}^{(n)}) - \hat{h} + \hat{z}_\theta^{(n)}\right) \otimes \left(h(\hat{v}^{(n)}) - \hat{h} + \hat{z}_\theta^{(n)}\right), \tag{4.27b}$$

where $\hat{w}_\theta^{(n)}$ and $\hat{z}_\theta^{(n)}$ are trainable correction terms that depend on the state $\hat{v}^{(n)}$, the observation $h(\hat{v}^{(n)})$, the true observation $y^\dagger$, and the ensemble $\{(\hat{v}^{(n)}, h(\hat{v}^{(n)}))\}_{n=1}^N$.

The correction terms are defined by a neural operator $\mathfrak{F}$ as in (2.24) for the mean-field perspective presented in Section 2. In our practical implementation, we replace the dependence on ρ_h in $\mathfrak{F}$ by the ensemble $\{(\hat{v}^{(\ell)}, h(\hat{v}^{(\ell)}))\}_{\ell=1}^N$ and use the set transformer F^{ST} (4.25) to process the ensemble into a feature vector f_v. Recall that this can be viewed as using the empirical approximation $\rho_h^{(N)} \approx \rho_h$ given by (4.2).

To learn the correction terms for each particle, we then train a neural network $F^{\mathrm{gain}} : \mathbb{R}^{d_v} \times \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \times \mathbb{R}^{d_{\mathrm{ST}}} \to \mathbb{R}^{d_v} \times \mathbb{R}^{d_y}$ with the standard multilayer perceptron (MLP) architecture that takes the feature vector for the ensemble as an input:

$$F^{\mathrm{gain}}\left(\hat{v}^{(n)}, h(\hat{v}^{(n)}), y^\dagger, f_v; \theta_{\mathrm{gain}}\right) = \left(\hat{w}_\theta^{(n)}, \hat{z}_\theta^{(n)}\right), \tag{4.28}$$

where θ_{gain} denotes trainable parameters.

From (4.27a) and (4.27b), it is evident that if the correction terms satisfy $\hat{w}_\theta^{(n)} = \hat{z}_\theta^{(n)} = 0$, the formulation reduces to the EnKF (2.16). When the dynamics and observation models are linear, or in other words, when the filtering distribution of the states v is Gaussian, the Kalman filter achieves optimality [7,8,48]. Therefore, the motivation for our proposed scheme (2.22) lies in its optimality in the linear (or Gaussian) setting, while further aiming to improve the performance of the Kalman filter in nonlinear problems by learning the correction terms.

Remark 11 (On the Invertibility in the Gain Computation)**.** The matrix $K_\theta^{(2)}$, defined in (4.27b) as an average of outer products, is positive semi-definite. Since Γ is a fixed positive definite matrix, the sum $K_\theta^{(2)} + \Gamma$ is also positive definite and thus invertible. Crucially, the minimum eigenvalue of $K_\theta^{(2)} + \Gamma$ is bounded from below by the minimum eigenvalue of Γ, which is a positive constant independent of any trainable parameters. This property ensures the numerical stability of computing $(K_\theta^{(2)} + \Gamma)^{-1}$ during backpropagation.

Remark 12 (Correction Approach Versus End-to-end Gain Learning)**.** In addition to the proposed approach with correction terms, we have implemented the end-to-end learning of the operator $\mathfrak{B}$ presented in Section 2.4 and learning the Kalman gain K_θ directly without any constraint on its form. We find that the end-to-end approach or directly learning K_θ results in unstable and slower learning. A poorly initialized or an unconstrained K_θ leads to substantial divergence of the resulting filtered trajectory away from the ground truth trajectory. The EnKF-like structure in (4.26b) with learnable correction terms provides better stability while maintaining adaptability to nonlinear dynamics.

4.2.2. Learning the inflation

Inflation is a technique used to increase the ensemble spread, mitigating the underestimation of analysis covariance caused by sampling errors in ensemble-based methods. Finite ensemble sizes often lead to sampling errors in the forecast covariance, causing the filter to overly trust forecasts and risk filter divergence. Inflation helps maintain the response of the filter to observations by counteracting these effects.

The most common form of inflation is multiplicative inflation, where the analysis ensemble $\{v^{(n)}\}_{n=1}^N$ that is calculated after the analysis step (4.1c) is modified as follows

$$v^{(n)} \leftarrow v^{(n)} + (\alpha - 1)\left(v^{(n)} - \frac{1}{N}\sum_{\ell=1}^{N} v^{(\ell)}\right), \tag{4.29}$$

where $\alpha \geq 1$ is the inflation parameter. In our learning framework, we replace the term proportional to $(\alpha - 1)$ by a learned correction term $\hat{u}_\theta^{(n)}$,

that is output by an MLP-based neural network $F^{\text{infl}} : \mathbb{R}^{d_v} \times \mathbb{R}^{d_{\text{ST}}} \to \mathbb{R}^{d_v}$ given by

$$F^{\text{infl}}\left(v^{(n)}, f_v; \theta_{\text{infl}}\right) = \hat{u}_\theta^{(n)}, \tag{4.30}$$

where $v^{(n)}$ is the estimated state given by (4.1c), and θ_{infl} denotes trainable parameters. The learned inflation step is processed after the analysis step (4.1c), resulting in the particle update:

$$v^{(n)} \leftarrow v^{(n)} + \hat{u}_\theta^{(n)}. \tag{4.31}$$

Remark 13 (Dependence and Implications of the Learned Inflation Term)**.** The common form of inflation is a function of the ensemble states, i.e., not the observations, in (4.29). Similarly, the learned inflation term $\hat{u}_\theta^{(n)}$ in (4.30) depends on the ensemble states $\hat{v}^{(n)}$ and the feature vector f_v, without information of the true observation $y^\dagger$ or the observation noise covariance Γ. We avoid the risk that the inflation term would evolve into an end-to-end correction term, ensuring the learning performed by other components in our framework remains meaningful and effective.

4.2.3. Learning the localization

In spatially extended systems, sampling errors can lead to inaccurate covariance estimation, particularly when the sample size is small. True correlations typically decay with distance, but spurious long-range correlations may arise, distorting the covariance structure. Localization addresses this issue by suppressing artificial correlations while preserving meaningful local correlations.

Localization is implemented by damping covariances in an empirical covariance matrix based on distance. This is typically achieved by applying a Hadamard product (elementwise product, denoted by $\circ$) between the empirical covariance matrix and a predefined localization weight matrix L. For the dynamic model (2.1) with the state in d_v-dimensional space (i.e., $v \in \mathbb{R}^{d_v}$), $L \in \mathbb{R}^{d_v \times d_v}$ is calculated by

$$(L)_{k\ell} = g(D_{k\ell}), \quad \forall k, \ell \in [d_v], \tag{4.32}$$

where $g : \mathbb{R} \to \mathbb{R}$ is a function that maps distance values to localization weights (e.g., the Gaspari–Cohn function [53]), and $D_{k\ell}$ is the distance between index k and ℓ. To incorporate the localization technique in the EnKF, the Kalman gain formula (2.11b) is modified as

$$K = \left(\hat{C}^{vh} \circ L^{vh}\right)\left(\hat{C}^{hh} \circ L^{hh} + \Gamma\right)^{-1}, \tag{4.33}$$

where $L^{vh} \in \mathbb{R}^{d_v \times d_y}$ and $L^{hh} \in \mathbb{R}^{d_v \times d_y}$ are localization weight matrices corresponding to $\hat{C}^{vh}$ and $\hat{C}^{hh}$, respectively.

In our learning-based formulation of localization we proceed as follows: we incorporate localization into our parameterized Kalman gain by modifying (4.26b) to take the form

$$K_\theta = \left(K_\theta^{(1)} \circ \lessdot^{(1)}\right)\left(K_\theta^{(2)} \circ \lessdot^{(2)} + \Gamma\right)^{-1}, \tag{4.34}$$

The localization weight matrices $\lessdot^{(1)}$ and $\lessdot^{(2)}$ follow the structure in (4.32), where g is replaced by a parameterized distance-to-weight function $g_\theta : \mathbb{R} \to \mathbb{R}$.

In practice, it is not necessary to learn the entire function g_θ since we only have a finite number of different distances, defined by the spatial grid for discretized PDEs, and time indices in the setting of ODEs. We denote the unique distance values in $\{D_{k\ell}\}_{k,\ell=1}^{d_v}$ as $\{D_1, D_2, \ldots, D_{N_D}\}$, where N_D is the number of distinct distances. Given a vector $\hat{g} = [\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_{N_D}] \in \mathbb{R}^{N_D}$, we can define the function g_θ by

$$g_\theta(D_i) = \hat{g}_i, \quad \forall i \in [N_D]. \tag{4.35}$$

We employ a MLP-based neural network $F^{\text{loc}} : \mathbb{R}^{d_{\text{ST}}} \to \mathbb{R}^{N_D}$ given by:

$$F^{\text{loc}}\left(f_v; \theta_{\text{loc}}\right) = \hat{g}. \tag{4.36}$$

The parameterized localization weight matrices $\lessdot^{(1)}$ and $\lessdot^{(2)}$ are calculated based on the output of this learned function g_θ. In C.2, we provide a step-by-step example illustrating the computation of the parameterized localization weight matrices.

Remark 14 (Implementation Insights for Learning Localization Weights)**.** Here we provide more details for the learning process of the localization weight matrices and make some comments on our proposed approach.

- The final layer of F^{loc} is a $\mathrm{softmax}$ function scaled by 2, ensuring all outputs lie within the interval $[0, 2]$. Although amplifying underestimated covariances is typically the responsibility of inflation alone, our architecture jointly performs localization and inflation. Thus, we allow the weights to range in $[0, 2]$ rather than $[0, 1]$ to grant additional flexibility in the localization component.
- Instead of learning a continuous parametric form for g_θ, we learn its values only at discrete distances $\{D_i\}_{i=1}^{N_D}$. This reduces model complexity while preserving localization expressivity.
- While EnKF-based localization methods typically use fixed weight matrices across time steps (e.g., using the Gaspari–Cohn function [53]), there is much interest in adaptive approaches [54]. Our learning-based approach adapts these matrices for each time step j using neural networks via its dependence on the ensemble. As shown in (4.36) (time index j omitted for simplicity), the network processes time-step-specific inputs, allowing the localization scheme to dynamically adjust based on the current system state.

Remark 15 (On sharing the distance-to-weight function across $L_\theta^{(1)}$ and $L_\theta^{(2)}$)**.** In forthcoming experiments in this paper, the observation operator h subsamples state components, so the distances between observation indices are induced by the same state-space metric used for state indices. Accordingly, we learn a single distance-to-weight function g_θ and use it consistently to build $L_\theta^{(1)}$ and $L_\theta^{(2)}$. For more general h (e.g., nonlinear or indirect observations), one may instead learn two distinct functions, $g_\theta^{(1)}$ and $g_\theta^{(2)}$, tailored to state-observation pairs for $L_\theta^{(1)}$ and observation-observation pairs for $L_\theta^{(2)}$, respectively.

4.3. Loss function

This section outlines the procedure for training the neural network parameters, including the gain, inflation and localization (Section 4.2). Based on the stochastic dynamic model (2.1) and the data observation model (2.2), we make the follow assumption for the training:

Data Assumption 1. *Our training is conducted for fixed dynamics Ψ, observation operator h, and noise covariance matrices Σ and Γ. The time step Δt is also fixed (and incorporated in Ψ). The initial condition is a Gaussian with a fixed covariance C_0, i.e. $v_0 \sim \mathsf{N}(\bullet, C_0)$. For some $M, J \in \mathbb{N}$, the training data consists of M trajectories of length $J+1$, extracted from a single long trajectory generated by propagating the dynamic model (2.1) for $M(J+1)$ steps. Each sub-trajectory $\{v_j^\dagger\}_{j=0}^J$ has corresponding observations $\{y_j^\dagger\}_{j=1}^J$ generated according to (2.2).*

For each trajectory, we generate estimated states from our ensemble-based data assimilation model to compare with the true states from the dynamic model, enabling us to train the parameters θ_{ST}, θ_{gain}, θ_{infl}, and θ_{loc}, for the set transformer F^{ST} (4.25), and MLPs F^{gain} (4.28), F^{infl} (4.30) and F^{loc} (4.36) respectively.

We initialize an ensemble $\{v_0^{(n)}\}_{n=1}^N$ at the initial time step $j=0$ for each sub-trajectory, with members sampled i.i.d. from a Gaussian distribution with the mean $v_0^{(n)}$ and the covariance C_0:

$$v_0^{(n)} \sim \mathsf{N}(v_0^\dagger, C_0), \quad n = 1, \dots, N. \tag{4.37}$$

For $j = 0, 1, \dots, J-1$ and trainable parameters $\theta = \{\theta_{\text{ST}}, \theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}\}$, we update the ensemble $\{v_j^{(n)}(\theta)\}_{n=1}^N$ to $\{v_{j+1}^{(n)}(\theta)\}_{n=1}^N$ according to

$$\hat{v}_{j+1}^{(n)}(\theta) = \Psi(v_j^{(n)}(\theta)) + \xi_j^{(n)}, \quad \xi_j^{(n)} \sim \mathsf{N}(0, \Sigma), \tag{4.38a}$$

$$\hat{y}_{j+1}^{(n)}(\theta) = h(\hat{v}_{j+1}^{(n)}(\theta)) + \eta_{j+1}^{(n)}, \quad \eta_{j+1}^{(n)} \sim \mathsf{N}(0, \Gamma), \tag{4.38b}$$

$$v_{j+1}^{(n)}(\theta) = (\hat{v}_{j+1}^{(n)}(\theta) + \hat{u}_\theta^{(n)}) + (K_\theta)_{j+1}\left(y_{j+1}^\dagger - \hat{y}_{j+1}^{(n)}(\theta)\right), \tag{4.38c}$$

where $(K_\theta)_{j+1}$ is our learned parameterized gain matrix, including localization, and $\hat{u}_\theta^{(n)}$ is the inflation term according to Section 4.2. For all parameters θ we choose $v_0^{(n)}(\theta) = v_0^{(n)}$. The ensembles $\{v_j^{(n)}(\theta)\}_{n=1}^N$ depend on the trainable parameters θ for $j = 1, 2, \dots, J$.

In this paper we focus on state estimation, given by the mean of the estimated ensemble $\{v_j^{(n)}(\theta)\}_{n=1}^N$, i.e.,

$$\bar{v}_j(\theta) = \frac{1}{N}\sum_{n=1}^N v_j^{(n)}(\theta). \tag{4.39}$$

To train our neural network parameters effectively, we introduce a loss function that quantifies the discrepancy between the estimated states and the true states across the mth sub-trajectory $V_m = \{v_j^\dagger\}_{j=1}^J$, given by:

$$\mathcal{L}_m(\theta) = \frac{1}{J}\sum_{j=1}^J \frac{\|\bar{v}_j(\theta) - v_j^\dagger\|_2^2}{\|v_j^\dagger\|_2^2}, \tag{4.40}$$

which measures the relative accuracy of our ensemble mean predictions compared to the true trajectory. Our loss (4.40) is a relative squared L_2 loss, which is preferred over the standard L_2 loss for two main reasons. First, using the squared L_2 norm avoids computing square roots, which can lead to numerical instabilities when calculating gradients, especially when the error approaches zero. Second, normalizing by the true state magnitude ($\|v_j^\dagger\|_2^2$) makes the loss scale-invariant across different state variables and trajectory segments, ensuring balanced parameter updates regardless of the absolute magnitudes of the state components. This relative formulation is particularly important in dynamical systems where state variables may span different orders of magnitude.

In the training process, the loss is averaged across M training trajectories indexed by $m \in [M]$,

$$\mathcal{L}_{[M]}(\theta) = \frac{1}{M} \sum_{m \in [M]} \mathcal{L}_m(\theta). \tag{4.41}$$

The trainable parameter θ is optimized based on the above-defined loss $\mathcal{L}_{[M]}$. In practice, the loss $\mathcal{L}_{[M]}$ is approximated by a mini-batch as a subset of $[M]$.

Remark 16 (Gradient Truncation in Sequential Learning)**.** Computing the loss (4.40) requires evaluating ensemble estimates $\{\{v_j^{(n)}(\theta)\}_{n=1}^N\}_{j=1}^J$ across the entire trajectory. For the ensemble at step j, this involves nested evaluations of K_θ up to j times. These nested evaluation arise from the sequential nature of the trajectory, where each step depends on the computing of the Kalman gain from the previous steps, requiring backpropagation through the entire computational history. With large trajectory lengths J, this causes slow training and numerical instability since it is nearly equivalent to training a very deep neural network.

To address this issue, we introduce a gradient-detach hyperparameter $J_0 \in \mathbb{N}$. This hyperparameter limits the gradient computation for the loss related to time step j to only the steps $j, j-1, \dots, j-J_0+1$. This approach assumes that the trajectory at time step j is not significantly affected by parameter changes in the Kalman gain for earlier time steps, which is reasonable given the diminishing influence of distant past states in many dynamical systems. This technique significantly improves training speed and stability.

Remark 17 (Training Targets and Probabilistic Losses)**.** In the loss (4.40), the ground-truth states used for supervised training are obtained by simulating the known dynamical model (forward integration of Ψ with prescribed noise model). Thus, the availability of training data hinges on access to an explicit and tractable dynamic model. In high-dimensional settings where such a model is unavailable or impractical, a surrogate simulator (e.g., a reduced-order or data-driven one) can be used to synthesize training trajectories and observations. Furthermore, while the proposed loss (4.40) focuses on state estimation, a promising direction for future work is to adopt probabilistic loss functions that target the full filtering distribution $\mathbb{P}(v_j \mid Y_j)$ rather than only the mean-based point estimator; see Section 1.1.

4.4. Fine-tuning inflation & localization

Following the developments in Section 4.2, we may apply a set transformer (4.25) to process ensembles of any size into fixed-dimensional feature vectors. This allows for pretraining our MNMEF with ensemble size N and perform inference with $N' \neq N$. However in classical EnKF formulations, hyperparameters such as inflation and localization depend on the ensemble size, and allowing for this dependence significantly enhances performance. Thus, we consider fine-tuning a subset of the parameters for inference at ensemble size $N' \neq N$. We now detail this methodology.

The architecture defined in Section 4.2 is based on trainable parameters θ_{ST}, for the set transformer F^{ST} (4.25), and θ_{gain}, θ_{infl}, and θ_{loc} for the MLPs F^{gain} (4.28), F^{infl} (4.30) and F^{loc} (4.36) respectively. For efficient fine-tuning when transitioning from ensemble size N to N', we freeze θ_{ST} and only fine-tune θ_{gain}, θ_{infl}, and θ_{loc}. This specific choice of fine-tuning is effective because: (1) freezing θ_{ST} significantly reduces computational costs, as the set transformer contains the majority of network parameters; (2) the remaining parameters, though small in number, measurably improve performance when they are adapted to the ensemble size.

The training loss for the mth sub-trajectory $V_m = \{v_j^\dagger\}_{j=1}^J$ is defined by (4.39), (4.40):

$$\mathcal{L}_m^{(N)}(\theta_{\text{ST}}, \theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}) = \frac{1}{J} \sum_{j=1}^{J} \frac{\|\bar{v}_j(\theta) - v_j^\dagger\|_2^2}{\|v_j^\dagger\|_2^2}, \tag{4.42}$$

where we add the superscript N to emphasize dependence on the ensemble size. In pretraining with size N we solve

$$\theta_{\text{ST}}^{(N)}, \theta_{\text{gain}}^{(N)}, \theta_{\text{infl}}^{(N)}, \theta_{\text{loc}}^{(N)} = \arg\min_{\theta_{\text{ST}}, \theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}} \mathcal{L}_{[M]}^{(N)}(\theta_{\text{ST}}, \theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}). \tag{4.43}$$

During fine-tuning for size $N' \neq N$, using the same training trajectories we solve

$$\theta_{\text{gain}}^{(N')}, \theta_{\text{infl}}^{(N')}, \theta_{\text{loc}}^{(N')} = \arg\min_{\theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}} \mathcal{L}_{[M]}^{(N')}(\theta_{\text{ST}}^{(N)}, \theta_{\text{gain}}, \theta_{\text{infl}}, \theta_{\text{loc}}) \tag{4.44}$$

with fixed $\theta_{\text{ST}}^{(N)}$. In practice, we use small mini-batches of $[M]$ to estimate the losses in (4.43) and (4.44).

With the updated parameters, we proceed with ensemble size N' using:

$$\text{Feature Vector:} \quad f_v^{(N')} = F^{\text{ST}}\left(\{(\hat{v}^{(n)}, h(\hat{v}^{(n)}))\}_{n=1}^{N'}; \theta_{\text{ST}}^{(N)}\right), \tag{4.45a}$$

$$\text{Gain:} \quad F^{\text{gain}}\left(\bullet, f_v^{(N')}; \theta_{\text{gain}}^{(N')}\right), \tag{4.45b}$$

$$\text{Inflation:} \quad F^{\text{infl}}\left(\bullet, f_v^{(N')}; \theta_{\text{infl}}^{(N')}\right), \tag{4.45c}$$

$$\text{Localization:} \quad F^{\text{loc}}\left(f_v^{(N')}; \theta_{\text{loc}}^{(N')}\right). \tag{4.45d}$$

Although $\theta_{\text{ST}}^{(N)}$ remains unchanged after the fine-tuning, the feature vector $f_v^{(N')}$ differs from the pretraining features f_v since the input ensemble size changes from N to N'.

Remark 18 (Efficient Training Strategy)**.** While the per-particle update in the analysis step has linear complexity with respect to the ensemble size N for fixed gain matrix, the overall cost of one forward pass is dominated by the attention operations used to compute correction terms. These attention blocks scale quadratically in N. In practice, we adopt an efficient training strategy by pretraining on smaller ensemble sizes N and applying fine-tuning for larger ensembles $N' > N$, which maintains comparable performance while substantially reducing computational cost. Further improvements are possible by employing localized or linear-attention variants that can reduce the scaling in N.

In Section 5.8, we demonstrate this fine-tuning method's effectiveness, achieving comparable performance to full parameter fine-tuning with significantly improved efficiency.

5. Numerical experiments

In this section we present numerical experiments to demonstrate the improved numerical benefit of our proposed method, MNMEF, in comparison with leading existing ensemble methods; the experiments also serve to validate the effectiveness of several specific details within our approach. First, in Sections 5.2, 5.3, and 5.4, we compare our method with several classical approaches across several nonlinear dynamical systems – Lorenz '96, Kuramoto–Sivashinsky, and Lorenz '63. In Section 5.5, we compare the computational time between our MNMEF and other benchmarks. In Section 5.6, we show the robustness of our method to the inherent randomness in test trajectories. In Section 5.7, we conduct experiments to study the learning-based inflation and localization methods proposed in Sections 4.2.2 and 4.2.3. Subsequently, in Section 5.8, we perform experiments to illustrate the efficiency and effectiveness of our proposed fine-tuning method, introduced in Section 4.4. The code for our method is publicly available[3] and can be used to reproduce all of our experiments.

We reiterate that all reported experiments are for nonlinear problems. Our methodology is based on learning corrections to the Kalman filter structure inherent in the EnKF. For nonlinear problems there is bias in the EnKF which can be effectively learned and corrected for, and this is the essence of our approach. For linear problems there is no bias in the mean-field limit (although there will still be a bias with a finite ensemble [55]), so our methodology would attempt to fit neural networks to what is largely noise. Without any regularization or early stopping, this can lead to filter divergence for large enough training epochs. C.6 reports a linear-Gaussian experiment to illustrate this.

5.1. Experimental set-up

In the following three Sections 5.2, 5.3, and 5.4, we compare our method against several benchmark approaches:

1. **Localized EnKF perturbed observation (EnKF)** [9,10]: This is the classic implementation of EnKF that employs a stochastic update approach, where the observations are perturbed by adding random noise samples drawn from the observation noise distribution.
2. **Ensemble square root filter (ESRF)** [11,56]: This method is a deterministic variant of the EnKF. It directly transforms the ensemble using matrix square root operations to match the second order statistics of the EnKF, but without requiring stochastic perturbations.
3. **Local ensemble transform Kalman filter (LETKF)** [15]: This method builds upon the Ensemble Transform Kalman Filter (ETKF) [57] by incorporating spatial localization. ETKF is also a deterministic version of EnKF. Under linear-Gaussian assumptions and in the mean-field limit, ESRF and ETKF are theoretically equivalent, but they differ in practical implementation.
4. **Iterative ensemble Kalman filter (IEnKF)** [16]: This method enhances the EnKF (perturbed observation) by incorporating a variational scheme to better handle strongly nonlinear systems. It iteratively solves a minimization problem between adjacent time steps, progressively refining the solution for the analysis step through multiple iterations. In our experiments, we set the maximum number of iterations to 10 and use a convergence tolerance of 10^{-5}.

Remark 19. Inflation and localization are important for DA in high-dimensional systems. In our experiments, all benchmark filters (EnKF, ESRF, LETKF, and IEnKF) employ post-analysis inflation as in (4.29). For the higher-dimensional Lorenz-96 (Section 5.2) and Kuramoto-Sivashinsky (Section 5.3) systems, we apply localization to the **EnKF** and **LETKF**. We employ the Gaspari–Cohn (GC) function [53] as the distance-to-weight function for the localization according to the DAPPER package [58]. We do *not* localize ESRF or IEnKF for the following reasons:

1. LETKF can be viewed as ETKF with localization, and ETKF is theoretically equivalent to ESRF in the mean-field/linear-Gaussian regime; adding localization to ESRF would thus be essentially redundant with LETKF.
2. For IEnKF, a localized implementation is omitted because it requires multiple iterations with the dynamic model integrations at each step, making a joint grid search over inflation and localization computationally prohibitive for Lorenz '96 and KS, and because without careful radius tuning the method exhibited numerical instability (frequent NaNs due to filter divergence).

The hyperparameters for inflation and localization are optimized separately for each ensemble size via grid search; detailed settings and results are reported in C.5.

[3] https://github.com/wispcarey/DALearning.

In addition to the methods described above, we evaluate our MNMEF approach at various ensemble sizes. Unless otherwise specified, our MNMEF is pretrained with ensemble size $N = 10$ as described in Section 4.3. We then examine our model's performance across different ensemble sizes $N \in \{5, 10, 15, 20, 40, 60, 100\}$ to demonstrate the generalizability of our approach.

Our evaluation includes two variants of our method:

1. **Pretrained MNMEF**: Trained once at $N = 10$ for 1000 epochs and applied directly to all ensemble sizes, demonstrating transfer capability across different ensemble configurations. This appears as **'Pretrain'** in our figures.
2. **Fine-tuned MNMEF**: Fine tune the pretrained model specific to each target ensemble size for 20 epochs, as detailed in Section 4.4. This appears as **'Tuned'** in our figures.

The intermediate feature dimensions in the set transformer for different dynamic models are provided in C.1. The detailed intiail conditions for different dynamic models are provided in C.3. The training hyperparameters including trajectory lengths, learning rates and batch sizes, are provided in C.4.

For a fair comparison, all benchmark methods (EnKF, LETKF, and IEnKF) are tuned by RMSE with respect to the true state $v_j^\dagger$. Specifically, hyperparameters for inflation and localization are selected via comprehensive grid searches at multiple ensemble sizes to minimize true-state RMSE. The explored parameter spaces and heatmaps of the grid searches are reported in C.5. This approach ensures that all benchmark methods achieve close to their optimal performance; the benchmarks thus present a robust test of our proposed methodology. Notably, we performed hyperparameter selection for the benchmarks directly on their test set performance, without using a validation set, which only strengthens the benchmark methods' advantage in our comparisons.

The performance of the methods is evaluated using the difference between the estimated state (i.e., the mean of particles) and the ground truth state over the entire trajectory. In the continuous-time setting, let $\mathbf{v}^\dagger(t) : [0, T] \to \mathbb{R}^{d_v}$ denote the ground truth trajectory between time 0 and T, and let $\mathbf{v}(t) : [0, T] \to \mathbb{R}^{d_v}$ denote the estimated state trajectory. Let $\|\cdot\|_2$ denote the Euclidean norm in $\mathbb{R}^{d_v}$. Then, the performance can be evaluated using the relative $L^1 := L^1([0, T]; \mathbb{R}^{d_v})$ error:

$$\mathcal{E}(\mathbf{v}, \mathbf{v}^\dagger) = \frac{\|\mathbf{v} - \mathbf{v}^\dagger\|_{L^1}}{\|\mathbf{v}^\dagger\|_{L^1}} = \frac{\int_0^T \|\mathbf{v}(t) - \mathbf{v}^\dagger(t)\|_2 dt}{\int_0^T \|\mathbf{v}^\dagger(t)\|_2 dt}. \tag{5.1}$$

In the discrete-time setting, we interpret the discrete time index j as time $t_j = j\Delta t$, where $\Delta t = T/J$ is the time step size. The estimated trajectory is given by the sequence $V^{(N)} = \{\bar{v}_j^{(N)}\}_{j=1}^J$ where $\bar{v}_j^{(N)} = \frac{1}{N}\sum_{n=1}^N v_j^{(n)}$, and the ground truth trajectory is given by $V^\dagger = \{v_j^\dagger\}_{j=1}^J$. The relative discrete L^1 error can then be defined as

$$\mathcal{E}(V^{(N)}, V^\dagger) = \frac{\sum_{j=1}^J \|\mathbf{v}(j\Delta t) - \mathbf{v}^\dagger(j\Delta t)\|_2 \, \Delta t}{\sum_{j=1}^J \|\mathbf{v}^\dagger(j\Delta t)\|_2 \, \Delta t} = \frac{\sum_{j=1}^J \|\bar{v}_j - v_j^\dagger\|_2}{\sum_{j=1}^J \|v_j^\dagger\|_2}. \tag{5.2}$$

This error metric can be interpreted as a form of relative root-mean-square error (RMSE). At each time step j, the quantity $\|\bar{v}_j - v_j^\dagger\|_2$ corresponds to the root-mean-square error over the d_v-dimensional state vector. The overall metric thus represents a time-averaged spatial RMSE, normalized by the magnitude of the ground truth trajectory. In what follows, we refer to this error $\mathcal{E}(V^{(N)}, V^\dagger)$ as the relative RMSE (R-RMSE).

For M_{test} trajectories $\{V_m^\dagger\}_{m=1}^{M_{\text{test}}}$ and their corresponding estimated trajectories $\{V_m^{(N)}\}_{m=1}^{M_{\text{test}}}$, the averaged R-RMSE $\bar{\mathcal{E}}$ is given by:

$$\bar{\mathcal{E}} = \frac{1}{M_{\text{test}}} \sum_{m=1}^{M_{\text{test}}} \mathcal{E}(V_m^{(N)}, V_m^\dagger). \tag{5.3}$$

In the experiments presented in the remainder of this section, we adopt the averaged R-RMSE $\bar{\mathcal{E}}$ as the primary evaluation metric, allowing us to ameliorate the effect of randomness inherent in each individual run and providing a more reliable assessment of the overall performance of different methods. Unless otherwise specified, the number of test trajectories is fixed at $M_{\text{test}} = 64$.

To further highlight the advantages of our method, we consider the relative improvement in terms of R-RMSE $\mathcal{E}_{\text{RI}}$ defined as:

$$\mathcal{E}_{\text{RI}} = \frac{\bar{\mathcal{E}}_{\text{benchmark}} - \bar{\mathcal{E}}_{\text{ours}}}{\bar{\mathcal{E}}_{\text{benchmark}}}. \tag{5.4}$$

This metric quantifies the percentage reduction in R-RMSE achieved by our method compared to the benchmark method. When $\mathcal{E}_{RI} < 0$, our method performs worse than the benchmark.

We apply the following general hyperparameter setting in all of our experiments unless otherwise specified. The dynamic operator Ψ in Eq. (2.1) is obtained by solving a differential equation using the fourth-order Runge–Kutta (RK4) [59] method. The time step Δt denotes the interval between consecutive observations. For accurate numerical integration of the dynamics, we perform multiple RK4 steps within each interval Δt, utilizing an integration step size smaller than Δt. The dynamic noise $\xi \sim \mathcal{N}(0, \Sigma)$ is typically set at $\Sigma = \sigma_v^2 I$ where I is the identity matrix and $\sigma_v = 10^{-3}$ or 0. For the observation noise, $\eta \sim \mathcal{N}(0, \Gamma)$, we define its covariance matrix as $\Gamma = \sigma_y^2 I$, where we set σ_y to either 0.7 or 1.0.

5.2. Lorenz '96

The Lorenz '96 model is a widely-used set of ordinary differential equations that possess features of large-scale atmospheric dynamics [60]. The model takes the form of a damped-driven linear system, with additional energy conserving quadratic nonlinearity that

Table 1
Lorenz '96 system settings.

Category	Values
Parameters	$F = 8$
States	$v = (u_1, u_2, \dots, u_{40}) \in \mathbb{R}^d, \quad d = 40$
Observations	$h(v) = (u_1, u_5, \dots, u_{33}, u_{37}) \in \mathbb{R}^{d_{\text{obs}}}, \quad d_{\text{obs}} = 10$
Time Step	Observation time step: $\Delta t = 0.15$; 5 RK4 integration steps: $\Delta t/5 = 0.03$

induces cyclic interactions (waves) among the components of the system; this makes it particularly valuable for studying atmospheric predictability. The system dynamics are governed by the following equations:

$$\frac{du_i}{dt} = (u_{i+1} - u_{i-2})u_{i-1} - u_i + F, \quad i = 1, 2, \dots, d. \tag{5.5}$$

The parameter settings for the Lorenz '96 system are shown in Table 1. The forcing parameter $F = 8$ places the Lorenz '96 system in a chaotic regime when $d = 40$.

In Fig. 1, we compare the performance of our pretrained MNMEF and fine-tuned MNMEF against four benchmark methods: EnKF, ESRF, LETKF, and IEnKF, on the Lorenz '96 system. The upper row of plots in Fig. 1 displays the R-RMSE (5.3) for observation noise levels $\sigma_y = 0.7$ (left plot) and $\sigma_y = 1.0$ (right plot). The lower row of plots in Fig. 1 shows the relative improvement $\mathcal{E}_{\text{RI}}$ (5.4) of our fine-tuned method as compared to EnKF, IEnKF, and LETKF for these respective noise levels.

We observe that our fine-tuned model consistently outperforms all benchmarks. Indeed, our fine-tuned method achieves substantial improvement for small ensemble sizes and maintains a notable advantage at larger ensemble sizes, outperforming LETKF by approximately 15–20 %. Our pretrained model, trained with $N = 10$ ensembles also behaves competitively, but performs slightly worse than LETKF for large ensemble size $40, 60, 100$.

To further illustrate the performance differences between our MNMEF and the best-performance benchmark LETKF, we provide visualizations in Figs. 2 and 3 for $N = 10$ and $\sigma_y = 1.0$. The visualization is on the last 100 time steps (from 1401 to 1500) from a test trajectory of length 1500 and $\Delta t = 0.15$. In both figures, our MNMEF approach is the one pretrained with the ensemble size $N = 10$.

Fig. 2 presents a comparison of the estimated state trajectory. Panel (a) shows the ground truth, and panel (b) shows the sparse observations. The subsequent rows compare MNMEF and LETKF. Visually, MNMEF displays smaller magnitudes for the absolute error (panel (d)). In addition, the ensemble spread (panel (e)) for MNMEF is more consistent than LETKF. Fig. 3 focuses on the estimation of two specific dimensions: one observed (Dimension 1, panel (a)) and one unobserved (Dimension 2, panel (b)). For the observed dimension, both MNMEF and LETKF perform comparably well, with their ensemble means closely following the ground

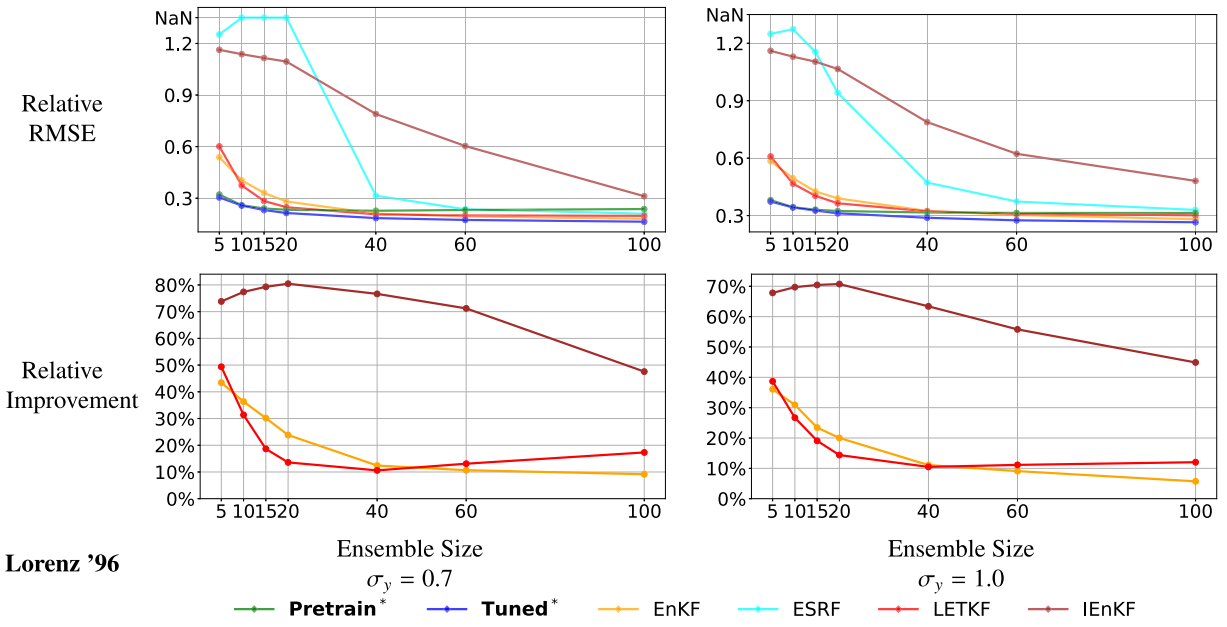


Fig. 1. Comparison results on the Lorenz '96 system. The upper row of plots shows direct R-RMSE comparisons between different methods, while the lower row illustrates the relative improvement of our fine-tuning method compared to benchmarks. These comparisons are presented for observation noise levels $\sigma_y = 0.7$ (left column) and $\sigma_y = 1.0$ (right column). Our proposed MNMEF method is highlighted in the legends with bold font and an asterisk (e.g. **Pretrain***). In summary, our fine-tuned MNMEF model consistently outperforms benchmarks, showing substantial improvements for small ensembles and a 15–20 % advantage over LETKF at larger ensemble sizes (eg. 60, 100).

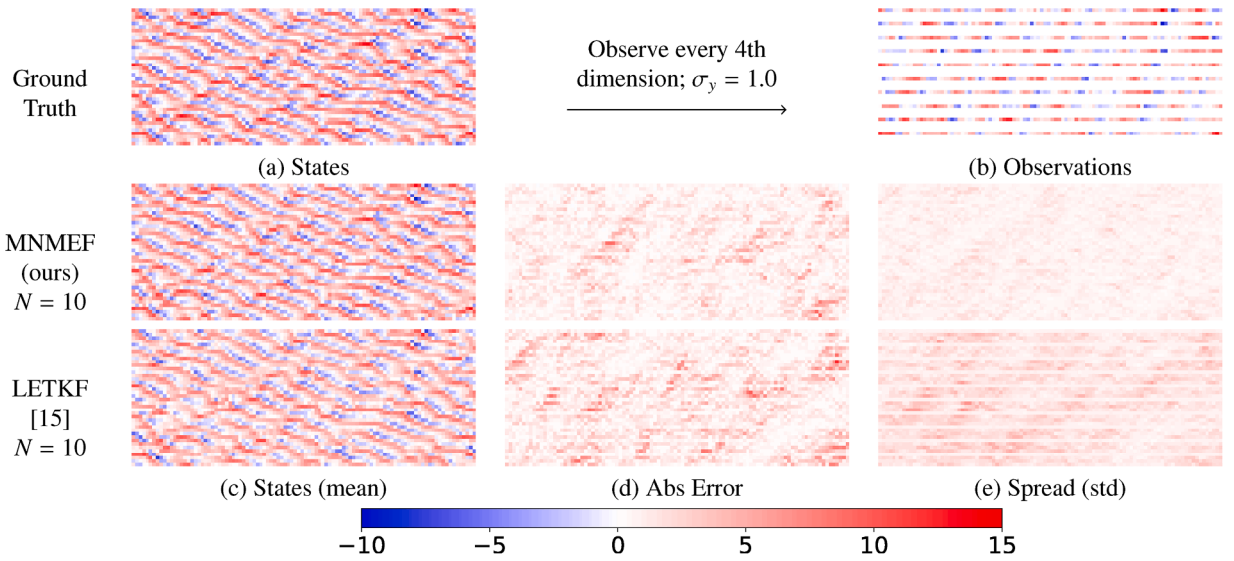


Fig. 2. Visualization of one test trajectory (time steps 1401–1500, $\Delta t = 0.15$) for Lorenz '96 states (vertical axis) over time (horizontal axis)with the observation noise $\sigma_y = 1.0$. Panel (a): Ground-truth states (unknown). Panel (b): Observations (known, every 4th dimension observed). Rows 2–3 show our method MNMEF pretrained on $N = 10$, and the benchmark LETKF with the ensemble size $N = 10$. Panel (c): state estimation (ensemble mean), Panel (d): absolute error of mean with respect to the ground truth, and Panel (e): ensemble spread (standard deviation).

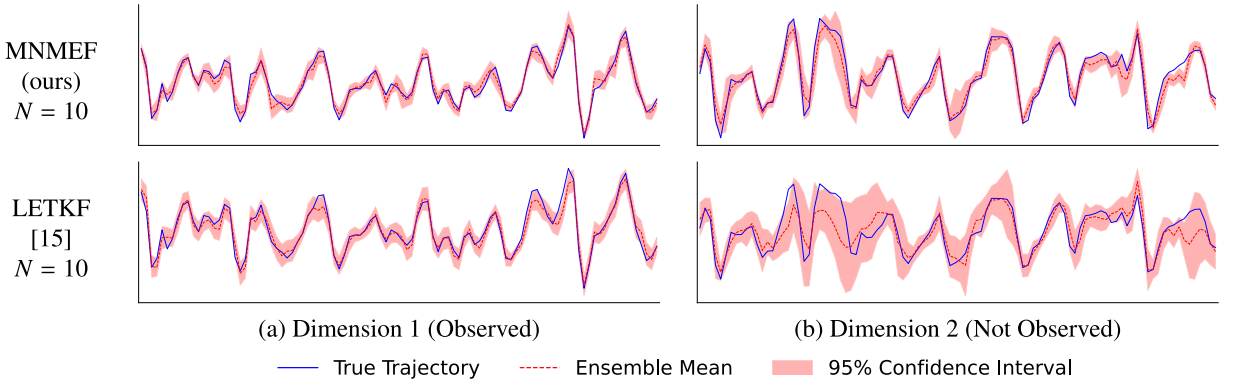


Fig. 3. Visualization of two dimensions (index 1 and 2) in one test trajectory (time steps 1401–1500, $\Delta t = 0.15$) for Lorenz '96 state values (vertical axis) over time (horizontal axis) with the ensemble size $N = 10$ and the observation noise $\sigma_y = 1.0$. Panel (a): Dimension 1, observed. Panel (b): Dimension 2, not observed. The first row is our method MNMEF, pretrained on $N = 10$, and the second row is the benchmark LETKF. The 95 % confidence intervals shown in figures are calculated as the ensemble mean $\pm$ 1.96 $\times$ the ensemble standard deviation. In summary, MNMEF performs significantly better on the unobserved dimension and comparably to LETKF on the observed dimension; meanwhile, MNMEF maintains an appropriate spread without suffering from filter degeneracy.

truth. However, for the unobserved dimension, MNMEF demonstrates a clear advantage in state estimation. Furthermore, MNMEF maintains an appropriate ensemble spread while avoiding filter degeneracy.

5.3. Kuramoto–Sivashinsky

The Kuramoto–Sivashinsky (KS) equation [61,62] is a widely studied nonlinear partial differential equation that describes the evolution of instabilities in spatially extended systems, capturing complex spatiotemporal chaotic dynamics arising in flame fronts, thin fluid films, and reaction-diffusion systems. Due to its chaotic behavior and sensitivity to initial conditions, the KS equation serves as a valuable testbed for developing and evaluating data assimilation algorithms. With the periodicity in space imposed to identity $x = L$ and $x = 0$, the one-dimensional KS equation for $u : [0, L] \times \mathbb{R}^+ \to \mathbb{R}$ takes the form

$$\frac{\partial u}{\partial t} + \frac{\partial^4 u}{\partial x^4} + \frac{\partial^2 u}{\partial x^2} + u\frac{\partial u}{\partial x} = 0, \quad (x, t) \in (0, L) \times \mathbb{R}^+, \tag{5.6a}$$

$$u(x, 0) = u_0, \quad \forall x \in [0, L]. \tag{5.6b}$$

Table 2
Kuramoto–Sivashinsky (KS) system settings.

Category	Values
Parameters	$L = 32\pi$
States	$x_j = jL/128, \quad j = 0, 1, \ldots, 127$
	$v = \big(u(x_0), u(x_1), \ldots, u(x_{127})\big) \in \mathbb{R}^d, \quad d = 128$
Observations	$h(v) = \big(u(x_0), u(x_8), \ldots, u(x_{120})\big) \in \mathbb{R}^{d_{\text{obs}}}, \quad d_{\text{obs}} = 16$
Time Step	Observation time step: $\Delta t = 1$; 4 ETDRK4 integration steps: $\Delta t/4 = 0.25$

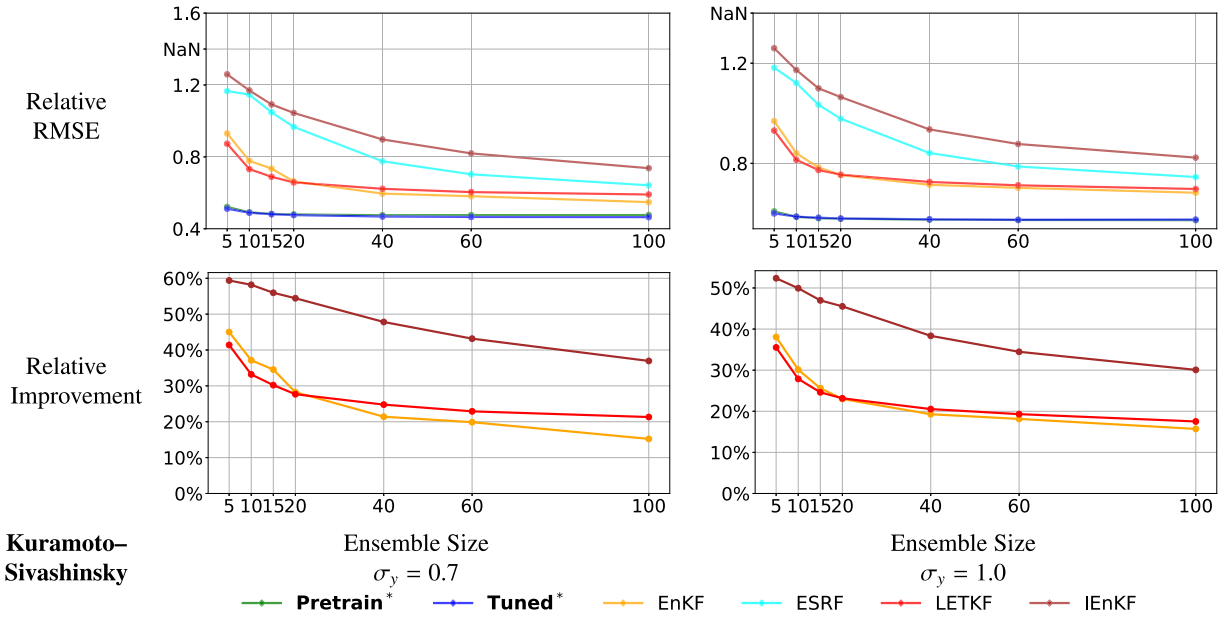


Fig. 4. Comparison results on the Kuramoto–Sivashinsky (KS) system. The upper row of plots shows direct R-RMSE comparisons between different methods, while the lower row illustrates the relative improvement of our fine-tuning method compared to benchmarks. These comparisons are presented for observation noise levels $\sigma_y = 0.7$ (left column) and $\sigma_y = 1.0$ (right column). Our proposed MNMEF method is highlighted in the legends with bold font and an asterisk (e.g. **Pretrain***). In summary, our fine-tuned MNMEF model consistently outperforms benchmarks, showing substantial improvements for small ensembles and around 20 % advantage over LETKF at larger sizes (eg. 60, 100).

The detailed experimental settings and parameters for the KS system are summarized in Table 2. Numerically, the KS Eq. (5.6) is discretized spatially using a Fourier pseudo-spectral method to handle periodic boundary conditions effectively. Time integration is performed with the exponential time-differencing fourth-order Runge-Kutta (ETDRK4) scheme [63], which advances the stiff linear operator analytically and is therefore not subject to the explicit linear Courant-Friedrichs-Lewy (CFL) restriction ($\Delta t \leq C\Delta x^4$) that would apply to fully explicit RK methods; the time step is chosen based on accuracy and nonlinear resolution rather than linear stability.

In Fig. 4, we compare the performance of our pretrained model and fine-tuned model against four benchmark methods: EnKF, ESRF, LETKF, and IEnKF, for the KS system with $\sigma_y = 0.7$ and 1.0, and provide the relative improvement of our fine-tuned method as compared to EnKF, IEnKF, and LETKF.

The comparison between our pretrained MNMEF and fine-tuned MNMEF methods and the benchmark methods, for the KS dynamical system, slightly differs from what we observed under the Lorenz '96 dynamics. In the KS setting, fine-tuning provides minimal additional benefit, with both our pretrained and fine-tuned MNMEF models performing nearly identically. Both variants significantly outperform benchmark methods across all ensemble sizes. The advantage is particularly pronounced at smaller ensemble sizes. Even at the largest ensemble size ($N = 100$), both our pretrained and fine-tuned methods still maintain approximately 20 % improvement over LETKF, the benchmark method with the best performance.

Similarly to the Lorenz '96 discussion in Section 5.2, we provide visualizations in Figs. 2 and 3 for $N = 10$ and $\sigma_y = 1.0$ to further illustrate the performance differences between our MNMEF and the best-performance benchmark LETKF on the KS system. The visualization is on the last 100 time steps (from 1901 to 2000) of a test trajectory with length 2000 and $\Delta t = 1$. In both figures, our MNMEF is the one pretrained with the ensemble size $N = 10$.

Fig. 5 presents a comparison of the estimated state trajectory. Fig. 6 visualizes the estimation of two specific dimensions: one observed (Dimension 1, panel (a)) and one unobserved (Dimension 2, panel (b)). In summary, our MNMEF performs slightly better

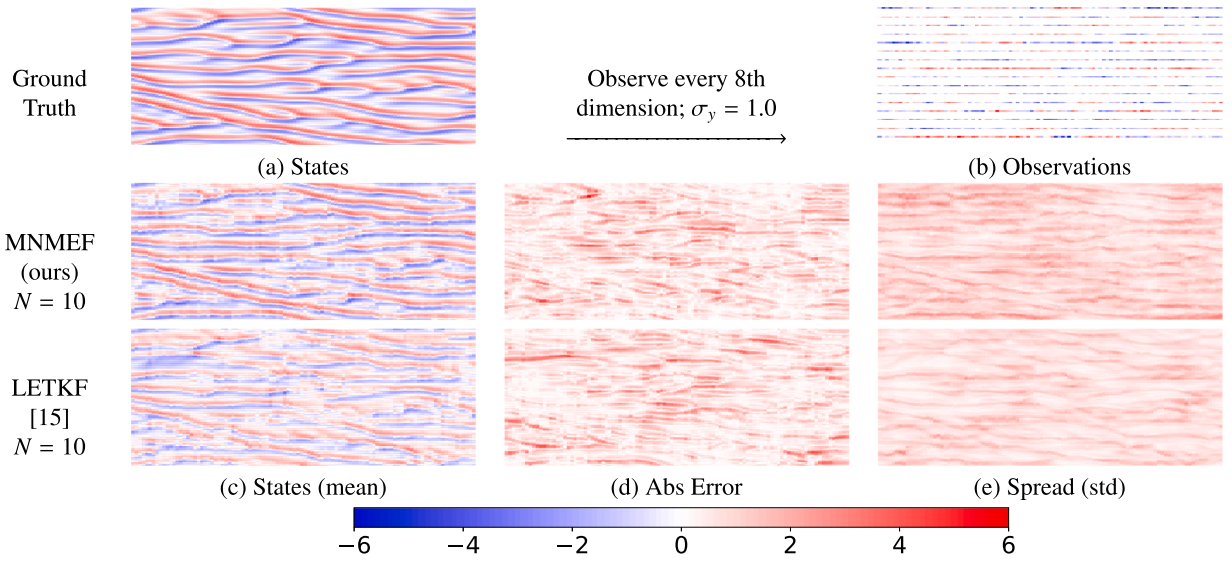


Fig. 5. Visualization of one test trajectory (time steps 1901–2000, $\Delta t = 1$) for Kuramoto–Sivashinshy (KS) states (vertical axis) over time (horizontal axis) with the observation noise $\sigma_y = 1.0$. Panel (a): Ground-truth states (unknown). Panel (b): Observations (known, every 8th dimension observed). Rows 2–3 show our method MNMEF pretrained on $N = 10$, and the benchmark LETKF with the ensemble size $N = 10$. Panel (c): state estimation (ensemble mean), Panel (d): absolute error with ground truth, and Panel (e): ensemble spread (standard deviation).

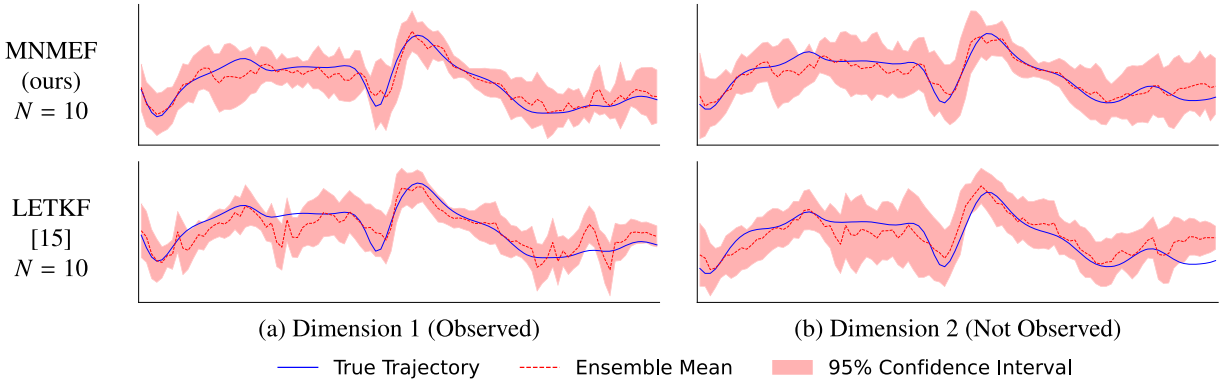


Fig. 6. Visualization of two dimensions (index 1 and 2) in one test trajectory (time steps 1901–2000, $\Delta t = 1$) for Kuramoto–Sivashinshy (KS) state values (vertical axis) over time (horizontal axis) with the ensemble size $N = 10$ and the observation noise $\sigma_y = 1.0$. Panel (a): Dimension 1, observed. Panel (b): Dimension 2, not observed. The first row is our method MNMEF, pretrained on $N = 10$, and the second row is the benchmark LETKF. The 95 % confidence intervals shown in figures are calculated as the ensemble mean $\pm$ 1.96 $\times$ the ensemble standard deviation. In summary, MNMEF performs slightly better on both the observed and the unobserved dimensions; meanwhile, MNMEF maintains an appropriate spread without suffering from filter degeneracy.

than LETKF on both the observed and the unobserved dimensions, which is verified by the R-RMSE quantitative comparison in Fig. 4. MNMEF maintains an appropriate ensemble spread, similarly to the LETKF.

Remark 20 (Implementation of Localization)**.** It is worth noting that methods other than LETKF could potentially accommodate localization techniques. For example, LETKF is essentially the ESRF combined with localization. However, since our implementation is based on the DAPPER package, which only implements localization for LETKF, we do not consider localization for the other four methods. Our results on Lorenz '96 and KS problems sufficiently demonstrate the advantages of our approach over other methods, even without localization. For larger ensemble sizes (e.g., $N = 60, 100$), where the benefits of localization diminish, our method still significantly outperforms the alternatives, like IEnKF.

5.4. Lorenz '63

The Lorenz '63 system is a simplified mathematical model for atmospheric convection [64], proposed prior to the Lorenz '96 model discussed in Section 5.2. It represents a highly simplified version of atmospheric dynamics with only three dimensions, described by

Table 3
Lorenz '63 system settings.

Category	Values
Parameters	$\sigma = 10, \quad \rho = 28, \quad \beta = \frac{8}{3}$
States	$v = (x, y, z) \in \mathbb{R}^d, \quad d = 3$
Observations	$h(v) = x \in \mathbb{R}^{d_{\text{obs}}}, \quad d_{\text{obs}} = 1$
Time Step	Observation time step: $\Delta t = 0.15$; 5 RK4 integration steps: $\Delta t/5 = 0.03$

the following set of ordinary differential equations:

$$\frac{dx}{dt} = \sigma(y - x), \tag{5.7a}$$

$$\frac{dy}{dt} = x(\rho - z) - y, \tag{5.7b}$$

$$\frac{dz}{dt} = xy - \beta z. \tag{5.7c}$$

The parameter settings for the Lorenz '63 system are shown in Table 3. The three parameters σ, ρ, and β represent the Prandtl number, Rayleigh number, and geometric factor, respectively, chosen by Lorenz based on simplified thermal convection experiments to produce chaotic dynamics characterized by sensitivity to initial conditions.

In Fig. 7, we compare the R-RMSE (5.3) performance of our pretrained MNMEF and fine-tuned MNMEF models against benchmark methods EnKF, ESRF, and IEnKF for the Lorenz '63 system. The upper row of plots in Fig. 7 displays these R-RMSE comparisons for observation noise levels $\sigma_y = 0.7$ (left plot) and $\sigma_y = 1.0$ (right plot). The LETKF method is not included in this comparison since the system has only three dimensions, making localization unnecessary. For a fair comparison, we turn off the localization part in our architecture by setting all entries in the localization weight matrices to be 1. We only fine-tune the parameters θ_{gain} for F^{gain} (4.28) and θ_{infl} for F^{infl} (4.30) for ensemble sizes $N \neq 10$. The lower row of plots in Fig. 7 shows the relative improvement $\mathcal{E}_{\text{RI}}$ (5.4) of our fine-tuned MNMEF method as compared to EnKF and IEnKF, for the respective noise levels shown.

We observe that the performance of different methods does not significantly change with increasing ensemble size. Our method consistently outperforms the EnKF and ESRF, but in the low-dimensional Lorenz '63 problem, it is outperformed by the IEnKF. As noted in Remark 21, IEnKF demonstrates strong performance in low-dimensional, highly nonlinear problems while struggling in high-dimensional systems without localization. This is consistent with our findings in the higher-dimensional experiments in Sections 5.2

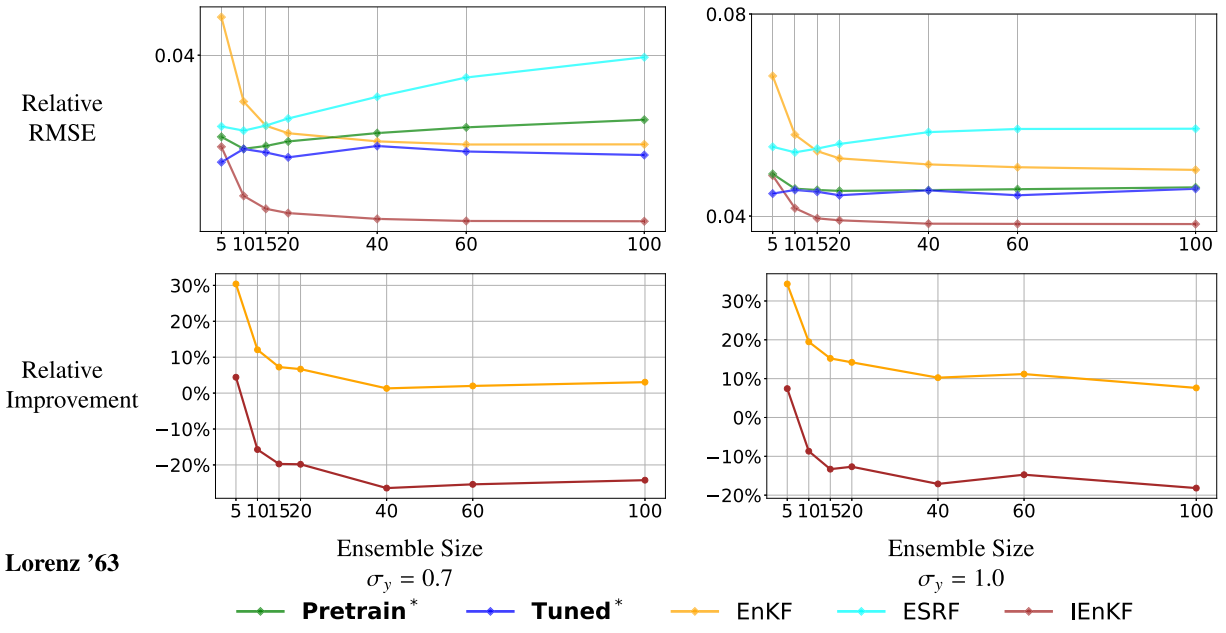


Fig. 7. Comparison results on the Lorenz '63 system. The upper row of plots shows direct R-RMSE comparisons between different methods, while the lower row illustrates the relative improvement of our fine-tuning method compared to benchmarks. These comparisons are presented for observation noise levels $\sigma_y = 0.7$ (left column) and $\sigma_y = 1.0$ (right column). Our proposed MNMEF method is highlighted in the legends with bold font and an asterisk (e.g. **Pretrain***).

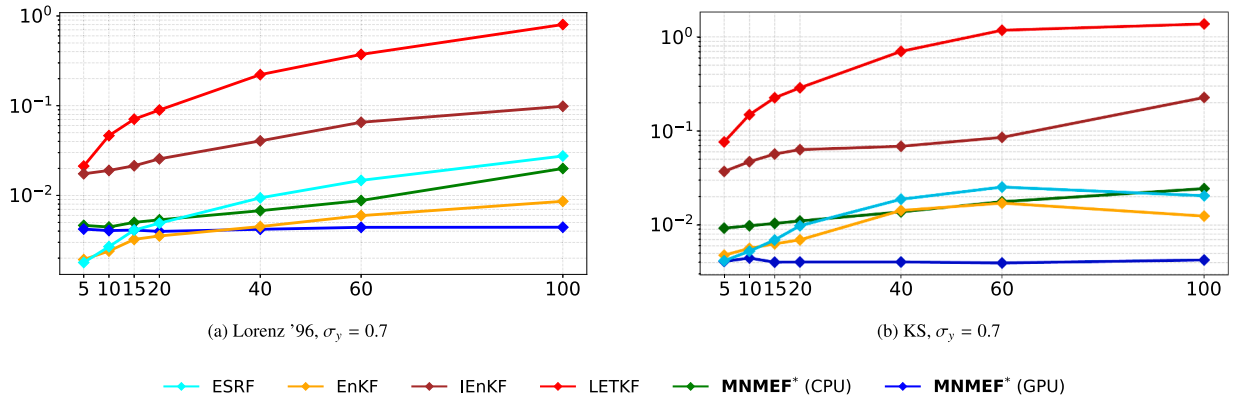


Fig. 8. Run time (s/analysis step), σ_y=0.7 only. All methods are timed on CPU (Intel(R) Core(TM) i9-14900KF). MNMEF is comparable in speed to EnKF and ESRF, and is slightly slower than EnKF because MNMEF additionally computes the correction terms. MNMEF remains much faster than LETKF and IEnKF. For MNMEF, we additionally report GPU timing on a single GPU (RTX 4080 Super), shown as the blue curve. The GPU version shows very stable runtime across these ensemble sizes due to efficient parallelization. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and 5.3, where our method substantially outperforms IEnKF. Furthermore, the stability analysis in Section 5.6 demonstrates that our method exhibits higher stability (lower standard deviation across test trajectories) than IEnKF.

Remark 21 (IEnKF Dimensionality Challenges)**.** The IEnKF [16] uses the same observational information as the standard EnKF, but reformulates the filtering problem as a lag-1 smoothing problem, and solves it iteratively to better handle nonlinearities. While the IEnKF demonstrates excellent performance in low-dimensional systems (e.g., the Lorenz '63 model, $d = 3$), its effectiveness diminishes as dimensionality increases (e.g., the Lorenz '96 model, $d = 40$). This deterioration occurs due to lack of localization in the implementation of the IEnKF we use; however, even with large ensembles with less need for localization, our method outperforms the IEnKF in the higher-dimensional systems. Moreover, IEnKF generally provides a substantial improvement over non-iterative methods only when the observations are sufficiently dense, as shown in [16] using experiments with the Lorenz '96 model.

5.5. Computational cost

In this subsection, we compare the run time per assimilation step (s/step) of MNMEF against benchmark methods (EnKF, ESRF, LETKF, and IEnKF) under matched CPU-only settings to ensure a fair comparison.

For an ensemble of size N, the per-step cost of MNMEF scales as $\mathcal{O}(N^2)$. The dominant terms are (i) the attention blocks in the set transformer, which entail pairwise interactions among N members, and (ii) the computation of the learned Kalman gain K_θ and its gradients, implemented via batched linear solves with $N \times N$ systems. We do not form matrix inverses explicitly. While classical baselines admit textbook complexities, their practical scaling depends on solver details, localization, and parallelization, so a full theoretical comparison is beyond the scope of this paper.

To reflect practical efficiency, we conduct a CPU-only head-to-head timing in a single Python environment. All CPU timings are run on Intel(R) Core(TM) i9-14900KF. We follow implementations in DAPPER [58] and provide our own CPU-based implementations of EnKF, ESRF, LETKF, and IEnKF. We deliberately avoid GPU ports for these baselines because methods such as LETKF and IEnKF do not map cleanly to efficient, large-batch vectorization on current accelerators.

We evaluate on Lorenz '96 and KS with the same configurations as in Sections 5.2 and 5.3. For each ensemble size N and observation noise level $\sigma_y = 0.7$, we report the mean of the per-step run time (s/analysis step), averaged over trajectories and time steps. We do not distinguish the pretrained and fine-tuned variants of our MNMEF since they share the same architecture. The set-transformer parameter counts follow Table 5. Results are visualized in Fig. 8. MNMEF is comparable in speed to EnKF and ESRF, and is slightly slower than EnKF because MNMEF additionally computes the correction terms. MNMEF remains much faster than LETKF and IEnKF.

For MNMEF, we provide an additional GPU-based run time in Fig. 8, since its original implementation is designed for GPU. MNMEF's additional GPU timings are obtained on a single NVIDIA RTX 4080 Super. The GPU version of MNMEF shows very stable runtime across these ensemble sizes due to efficient parallelization.

5.6. Robustness to inherent randomness

Due to the inherent randomness in both the filtering problem and the algorithm, the same method may exhibit varying performance across different test trajectories. To mitigate this randomness and obtain a more reliable assessment of each method's effectiveness, we evaluate their performance on M_{test} trajectories and report the mean R-RMSE $\bar{\mathcal{E}}$ (5.3). While the mean R-RMSE provides a measure of overall accuracy, we are interested in the stability of each method across different trajectories. To quantify this stability, we compute

Table 4
Standard deviation (std) of the relative root mean square error (R-RMSE) as defined in (5.8). We consider both **Pretrain*** and **Tuned*** variants of our MNMEF method. The std shown in the table is an averaged std over ensemble sizes $N = 5, 10, 15, 20, 40, 60, 100$. A smaller std indicates better stability of the method. The lowest std in each column is highlighted in bold. The stds for both the pretrained and fine-tuned variants of our MNMEF method are similar and consistently lower than those of the benchmark methods, indicating greater robustness of our approach to the inherent randomness across different test trajectories.

Method	Lorenz '96 (5.5)		KS (5.6)		Lorenz '63 (5.7)	
	$\sigma_y = 1.0$	$\sigma_y = 0.7$	$\sigma_y = 1.0$	$\sigma_y = 0.7$	$\sigma_y = 1.0$	$\sigma_y = 0.7$
Pretrain*	**1.03e-2**	1.26e-2	**8.60e-3**	1.07e-2	**1.78e-3**	1.40e-3
Tuned*	1.08e-2	**1.12e-2**	8.91e-3	**1.07e-2**	1.79e-3	**1.29e-3**
EnKF [8]	2.16e-2	3.10e-2	2.66e-2	3.15e-2	3.41e-3	2.32e-3
ESRF [11]	3.09e-2	1.95e-2	2.59e-2	3.27e-2	2.78e-3	2.01e-3
LETKF [15]	2.63e-2	6.13e-2	2.66e-2	3.58e-2	–	–
IEnKF [16]	1.83e-2	2.61e-2	2.23e-2	3.03e-2	2.15e-3	1.44e-3

the standard deviation (std) of the R-RMSE values across all test trajectories $\{V_m^\dagger\}_{m=1}^{M_{\text{test}}}$ and their corresponding estimated trajectories $\{V_m\}_{m=1}^{M_{\text{test}}}$, defined as:

$$\sigma_{\mathcal{E}} = \sqrt{\frac{1}{M_{\text{test}}} \sum_{m=1}^{M_{\text{test}}} \left(\mathcal{E}(V_m, V_m^\dagger) - \bar{\mathcal{E}} \right)^2} \tag{5.8}$$

A smaller $\sigma_{\mathcal{E}}$ indicates better stability of the method. Therefore, in this section, we consider methods with lower std values to be more stable and thus preferable.

Remark 22. In ensemble-based data assimilation, variance- or standard-deviation-type quantities are commonly used to characterize ensemble spread [6]. In our work, the spread is visualized in Figs. 3 and 6 via the shaded bands: the 95% confidence interval is provided as the ensemble mean $\pm 1.96\times$ ensemble standard deviation. Our focus here is state estimation rather than full uncertainty calibration, so we do not present a quantitative comparison of spread metrics. Note that the stability measure $\sigma_{\mathcal{E}}$ in (5.8) serves a different purpose: it assesses the consistency of estimation errors across test trajectories, not the dispersion of ensemble members.

In Table 4, we show the std of different methods on the dataset Lorenz '96, KS and Lorenz '63 with the observation noise $\sigma_y = 1.0, 0.7$. The std values in the table is the averaged std over the ensemble sizes $N = 5, 10, 15, 20, 40, 60, 100$. Based on Table 4, we observe that our pretrained MNMEF and fine-tuned MNMEF demonstrate comparable standard deviation values, both of which are significantly lower than all benchmark methods. This indicates that our approaches maintain more consistent performance across different test trajectories. The enhanced stability suggests that our methods are more reliable in practical applications.

5.7. Inflation and localization

In this section, we will conduct further experiments on learning the inflation (Section 4.2.2) and localization (Section 4.2.3) components in the architecture of our MNMEF method. These experiments demonstrate that what our method learns has many similarities with what is used in the application of inflation and localization techniques to the benchmark methods.

5.7.1. Inflation experiments

In Section 4.2.2, we introduce our learning-based inflation scheme (4.31) which depends on the learned output $\hat{u}_\theta^{(n)}$ from the neural network F^{infl} (4.30).

We are particularly interested in quantifying the effect of our proposed inflation scheme on the overall performance of our method. Additionally, despite not providing the inflation term $\hat{u}_\theta^{(n)}$ with information about the true observation $y^\dagger$ or observation noise Γ in F^{infl}, $\hat{u}_\theta^{(n)}$ has significantly more degrees of freedom than classical inflation approaches. This raises concerns that $\hat{u}_\theta^{(n)}$ might exceed the scope of inflation: that it might not act merely as a correction term but might completely overwhelm the updates, thus rendering other parts of our architecture ineffective. We show that this is not the case, but at the same time we showcase the benefits of including this learned inflation-like correction.

For our experimental approach, we first pretrain the models and then fine-tune them separately for various ensemble sizes. During inference, we deliberately disable the learned inflation by setting $\hat{u}_\theta^{(n)} = 0$ and analyze the resulting change in the average R-RMSE $\bar{\mathcal{E}}$ (5.3). We stress that this manipulation does not represent an optimal or even well-trained MNMEF filter without inflation; instead, it intentionally removes the correction term to create a more challenging robustness test-probing whether the filter remains stable and avoids divergence in the absence of inflation. These experiments are conducted on the KS dynamical system (5.6) only.

Our results are presented in Fig. 9. We compare the mean R-RMSE $\bar{\mathcal{E}}$ between our standard architecture with the trained inflation (blue curve) and the no inflation version by setting $\hat{u}_\theta^{(n)} = 0$ (red curve) for observation noise $\sigma_y = 0.7$ and 1.0. The results in Fig. 9 clearly demonstrate the beneficial impact of our trained inflation approach. In addition, our method remains functional, with increased R-RMSE values, when we remove inflation by setting $\hat{u}_\theta^{(n)} = 0$ (equivalent to $\alpha = 1$ in the EnKF). This confirms that our experiment,

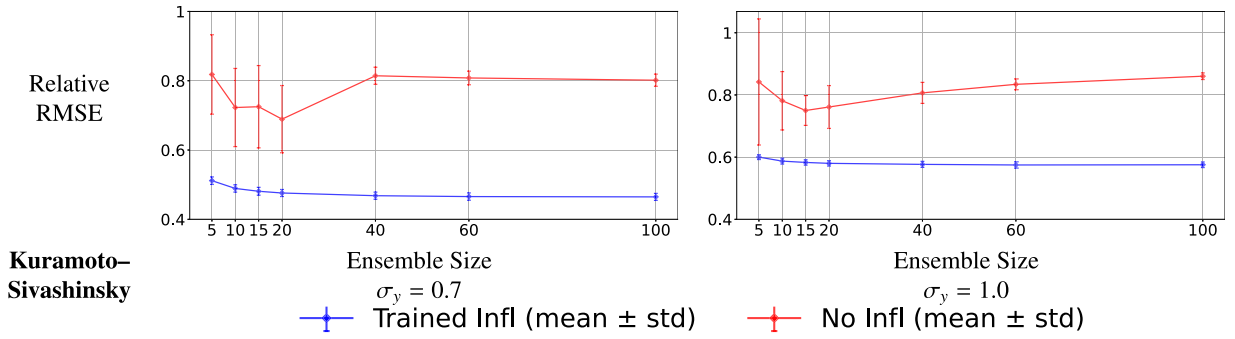


Fig. 9. Comparison of the trained inflation versus the no inflation scenario when applied to the KS system (5.6), using our MNMEF method, fine-tuned for different ensemble sizes. The plots compare R-RMSE with and without inflation at observation noise levels $\sigma_y = 0.7$ (left plot) and $\sigma_y = 1.0$ (right plot). In summary, our trained inflation effectively improves the performance, but even without inflation, our MNMEF does not completely fail.

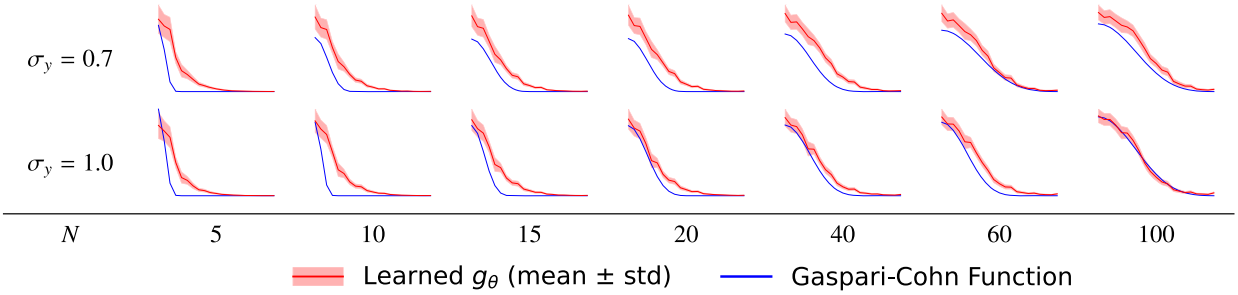


Fig. 10. Comparison of our learned distance-to-weight function g_θ (red, mean ± 1 std across different time steps due to its adaptivity) with the LETKF Gaspari–Cohn (GC) function (blue) on the Lorenz '96 model for $\sigma_y = 1.0, 0.7$ and ensemble sizes $N = 5, 10, 15, 20, 40, 60, 100$. The learned distance-to-weight function is from our MNMEF pretrained on $N = 10$ and fine-tuned for different ensemble sizes. The LETKF GC radius parameters are optimally selected via grid search. Remarkably, even without any explicit constraints on the shape of g_θ, the learned weights naturally decrease as the distance increases, similar to the empirical GC function in LETKF. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

while intentionally suboptimal, demonstrates that the proposed MNMEF remains numerically stable and does not diverge even in this deliberately more difficult configuration. This addresses our concern, because, if inflation were dominating the architecture by masking the contribution of the state $v^{(n)}$, removing it would cause model collapse.

5.7.2. *Localization experiments*

According to Section 4.2.3, our learning-based localization approach essentially learns a parameterized function g_θ, which maps distance to weight values. During training, we only enforce the constraint that the output of this function must lie within the interval [0,2], i.e., $g_\theta : \mathbb{R} \to [0, 2]$. Unlike traditional localization functions used in EnKF such as the Gaspari-Cohn function, our approach does not impose any explicit constraints on the shape or monotonicity of g_θ.

In this subsection, we present experimental comparisons between our learned function g_θ and the Gaspari-Cohn (GC) function with the optimal localization radius, found by grid-search, for LETKF; see Fig. 10. For ensemble sizes $N = 5, 10, 15, 20, 40, 60, 100$, the optimal GC radius values chosen are $1, 1, 2, 3, 4, 4, 6$ (multiplied by $\sqrt{10/3}$ when implemented [58]), respectively. The learned function g_θ (shown as red curves with the mean and standard deviation of the learned g_θ across assimilation time steps) demonstrates a similar shape to the GC function with optimally chosen radius (blue curves). Notably, our approach dynamically adjusts the localization weights at each assimilation step based on ensemble size and distribution, while the GC function remains static throughout the assimilation process.

The results in Fig. 10 demostrates the effectiveness of our learned localization. Despite imposing no explicit constraints on shape or monotonicity, the learned function g_θ naturally exhibits similar behavior to the GC function: it decreases monotonically with distance and shares a similar shape. A key difference is that our function occasionally exceeds values of 1 at short distances, reflecting a combined localization and inflation effect where covariance between closely spaced dimensions is amplified, since we are learning the inflation and localization simultaneously. This adaptivity allows g_θ to dynamically adjust to the ensemble distribution, leading to improved performance over the fixed GC function, as supported by RMSE results in Sections 5.2 and 5.3.

Remark 23 (Using GC localization directly)**.** Experimental results suggest that one can use the GC function to avoid training the localization. However, this approach still requires selecting the radius hyperparameter and typically yields inferior performance compared to our learned localization.

Table 5
Comparison of total parameters, fine-tuning parameters, and epochs for various datasets.

Dataset	Total Params	FT Params	Pretrain Epochs	FT Epochs
Lorenz '96	341,551	63,343	1000	20
KS	374,289	90,065	1000	20
Lorenz '63	309,383	34,119	1000	20

Table 6
Comparison of pretraining and fine-tuning computational time for observation noise $\sigma_y = 1.0$ with different ensemble sizes on a single RTX 4080 Super GPU. The ratio is between the computation time of the fine-tuning stage and the time of the pretraining stage. The fine-tuning time increases with the ensemble size, but it remains significantly faster compared to the pretraining stage.

Ens Size	$N = 10$	$N' = 20$		$N' = 40$		$N' = 100$	
Dataset	Pretrain (hrs)	FT (hrs)	Ratio	FT (hrs)	Ratio	FT (hrs)	Ratio
Lorenz '96	4.03	0.051	1.27 %	0.065	1.61 %	0.133	3.30 %
KS	8.22	0.106	1.29 %	0.132	1.61 %	0.260	3.16 %
Lorenz '63	1.82	0.022	1.21 %	0.038	2.09 %	0.086	4.73 %

5.8. *Fine-tuning for different ensemble size*

According to Section 4.4, our proposed fine-tuning strategy freezes parameters θ_{ST} of the set transformer F^{ST} (4.25), while updating parameters θ_{gain}, θ_{infl} and θ_{loc} of the MLPs that control the gain (4.28), inflation (4.30), and localization (4.36). Here we conduct experiments to illustrate the efficiency and effectiveness of our proposed fine-tuning approach.

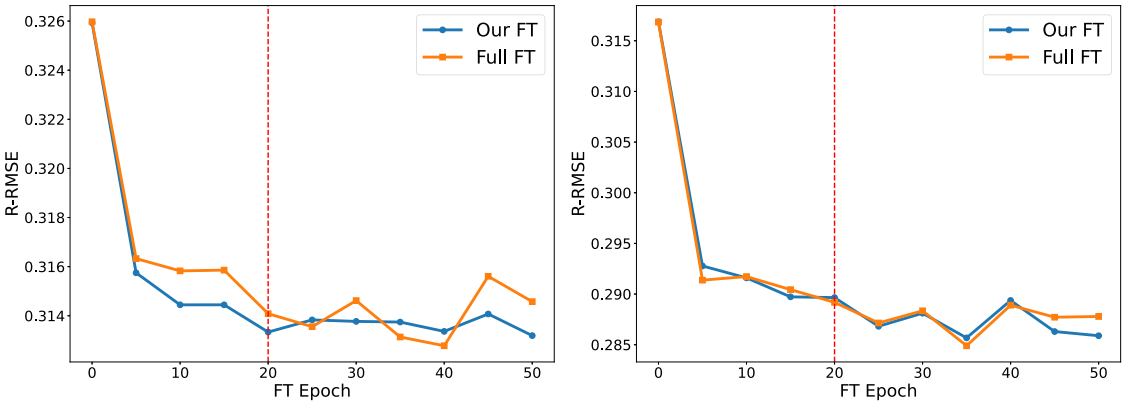
Firstly, we consider the time complexity between the pretraining stage and fine-tuning stage. Let P_{pretrain} and P_{ft} denote the total number of pretraining and fine-tuning parameters respectively, i.e.

$$P_{\text{pretrain}} = |\theta_{\text{ST}}| + |\theta_{\text{gain}}| + |\theta_{\text{infl}}| + |\theta_{\text{loc}}|, \tag{5.9a}$$

$$P_{\text{ft}} = |\theta_{\text{gain}}| + |\theta_{\text{infl}}| + |\theta_{\text{loc}}|. \tag{5.9b}$$

For fixed dynamic and training trajectories, the time complexity for pretraining with the ensemble size N is $\mathcal{O}(ENP_t)$ while the fine-tuning with the ensemble size N' has the time complexity $\mathcal{O}(E'N'P_{ft})$, where E and E' are the epochs for pretraining and fine-tuning, respectively.

We provide a detailed comparison in Table 5, presenting the total number of pretraining parameters P_{pretrain}, fine-tuning parameters P_{ft}, and the number of epochs E used during the pretraining and fine-tuning phases. When the ensemble size during pretraining (N) is close to the ensemble size used in fine-tuning (N'), Table 5 shows that the fine-tuning stage requires less than 1 % of the pretraining computational time. To quantitatively support this observation, we present detailed computational time comparisons in Table 6. All training times reported here are obtained using a single RTX 4080 Super GPU. The training batch sizes are adaptively chosen to fully use the 16 GB memory.



(a) Ensemble Size 20 (b) Ensemble Size 40

Fig. 11. Comparison of R-RMSE across training epochs on the Lorenz '96 model for two fine-tuning strategies under two ensemble sizes, (a) 20 and (b) 40. The vertical dashed red line indicates the stopping epoch of our proposed fine-tuning approach. Results demonstrate that our fine-tuning on a small subset of parameters achieves comparable RMSE performance to fine-tuning all parameters. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Combining the information from Tables 5 and 6, we observe that our fine-tuning strategy demonstrates remarkable efficiency compared to the pretraining stage. As detailed in Sections 5.4–5.3, this highly efficient fine-tuning significantly improves the model's performance, particularly when the ensemble sizes for inference and pretraining differ substantially.

To further validate the effectiveness of our proposed fine-tuning approach, we perform additional experiments comparing our method (Section 4.4) against alternative fine-tuning strategies. Specifically, we investigate and quantitatively compare the RMSE performance of the following fine-tuning scenarios: (1) fine-tuning all model parameters and (2) fine-tuning for a longer period (i.e., more epochs).

Fig. 11 shows RMSE results from fine-tuning experiments on the Lorenz '96 model with ensemble sizes $N' = 20, 40$, using a model pretrained with ensemble size $N = 10$. We compare our proposed method from Section 4.4 against fine-tuning all parameters, both running up to 50 epochs. The results demonstrate that both approaches achieve nearly identical RMSE performance, with minimal improvement observed after 20 epochs. Combined with our efficiency analysis in Tables 5 and 6, these findings confirm that fine-tuning all parameters is unnecessary, and that 20 epochs is sufficient for the near optimal performance.

6. Conclusions

In this paper, we propose a novel machine learning-based approach, MNMEF, for state estimation in data assimilation. Our method features a scheme which subsumes the ensemble Kalman filter (EnKF), as a special case, ensuring optimality for linear, Gaussian problems, while extending beyond Gaussian applications through our learnable correction terms. A key advantage of our method is its ability to be trained at one ensemble size and then directly applied, possibly with a cheap fine-tuning of a small subset of parameters, for different ensemble sizes. This property is a consequence of the mean-field interpretation of the set transformer, the machine learning architecture underlying the methodology. Indeed, in this paper we introduce neural operators acting on the metric space of probability measures, which we call *measure neural mappings* (MNM). We generalize transformers to this setting and show how the set transformer itself may be viewed as a particle approximation of such a mean-field model.

Our method adopts a form similar to the Kalman filter in the mean-field perspective, with the core innovation being a parameterized gain matrix analogous to the Kalman gain (Section 2). Since this parameterized gain matrix depends on the current filtering distribution, we employ the set transformer neural network architecture to process such measure inputs (Section 3). In practical applications, our method operates on an ensemble of particles (viewed as an empirical measure) and incorporates learnable mechanisms analogous to inflation and localization techniques in EnKF-based methods, enhancing performance with smaller ensemble sizes (Section 4). Our experiments across multiple challenging systems (Lorenz '63, Lorenz '96, and Kuramoto-Sivashinsky) demonstrate that our proposed method consistently outperforms classical approaches including the local ensemble transform Kalman filter (LETKF), with relative improvements of 15–30 % for most scenarios (Section 5).

Despite the promising results, our method has limitations and several directions for future improvement. Our current approach is limited by the loss function design, which primarily addresses state estimation rather than general filtering problems. Future work will consider probabilistic loss functions that can be used to estimate the filtering distribution.

Additionally, we plan to extend our approach beyond EnKF to include other filtering schemes, such as the ensemble square root filter (ESRF), which is a deterministic version of the EnKF, or variational methods such as iterative EnKF (iEnKF). The flexibility of our approach stems from the proposed MNM-based learning architecture, which can optimize variables related to the empirical distributions represented by ensembles.

We currently formulate the transformer as a neural operator accepting probability measures as inputs. In future work, we aim to enhance both the architecture and training methodology to simultaneously handle measure and function valued inputs, enabling amortized neural operators that generalize across different dynamics and observation models. Establishing universal approximation results and statistical guarantees for this formulation will also be of interest to further strengthen the theoretical foundations of our approach.

In conclusion, our work advances the field of data assimilation by introducing a novel machine learning approach, which is efficient for deployment, requiring only one pretraining stage and lightweight fine-tuning. Beyond immediate applications in state estimation, our theoretical contribution of extending attention mechanisms to operate as measure-to-measure transformations opens new research directions across multiple disciplines.

Data availability

I've shared the data / code in this article as a GitHub repository.

CRediT authorship contribution statement

Eviatar Bach: Writing – review & editing, Methodology, Formal analysis, Conceptualization; **Ricardo Baptista:** Writing – review & editing, Methodology, Formal analysis, Conceptualization; **Edoardo Calvello:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization; **Bohan Chen:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization; **Andrew Stuart:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interests

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Andrew Stuart reports financial support was provided by US Department of Defense. Andrew Stuart reports financial support was provided by National Science Foundation. Andrew Stuart reports financial support was provided by California Institute of Technology Resnick Sustainability Institute. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors are grateful to Arnaud Doucet for pointing them to work on the set transformer. The authors acknowledge support from a Department of Defense (DoD) Vannevar Bush Faculty Fellowship (award N00014-22-1-2790), NSF award AGS1835860 and from the Resnick Sustainability Institute; all support is held by AMS.

Appendix A. Overview of the architecture and our complete learning framework

In this appendix, we provide an overview of our complete learning framework. Our architecture (Section 4.2) is designed to handle varying ensemble sizes efficiently while maintaining high performance through a combination of neural network-based corrections and fine-tuning strategies. The training process follows the methodology described in Section 4.3, where the model parameters are initially trained on a fixed ensemble size N. When our approach is applied to a different ensemble size $N' \neq N$, we perform fine-tuning using the procedure detailed in Section 4.4.

Recall that in both the initial training and the fine-tuning process, we always work under Data Assumption 1: we fix dynamic operator Ψ (time step Δt implicitly embedded), observation operator h, and noise covariance matrices Σ and Γ. We generate multiple trajectories of the state-observation system through independent realizations of the driving noise sequences.

The core of our framework (Algorithm 1) is to update the ensemble from $\{v_j^{(n)}\}_{n=1}^N$ to $\{v_{j+1}^{(n)}\}_{n=1}^N$. This is a basic component for both the training and inference. Based on this, we can calculate the loss and do the initial training with the ensemble size N according to

Algorithm 1 One step evolution of the ensemble.

Require: Parameters $\theta = \{\theta_{\mathrm{ST}}, \theta_{\mathrm{gain}}, \theta_{\mathrm{infl}}, \theta_{\mathrm{loc}}\}$; current ensemble $\{v_j^{(n)}\}_{n=1}^N$.

Ensure: Ensemble for the next time step $\{v_{j+1}^{(n)}\}_{n=1}^N$.

1: Get predicted states $\{\hat{v}_{j+1}^{(n)}\}_{n=1}^N$ from $\{v_j^{(n)}\}_{n=1}^N$ (4.38a).

2: Get predicted observations $\{\hat{y}_{j+1}^{(n)}\}_{n=1}^N$ from $\{\hat{v}_{j+1}^{(n)}\}_{n=1}^N$ (4.38b).

3: Process the ensemble into a feature vector according to (4.25):

$$f_v = F^{\mathrm{ST}}\left(\{(\hat{v}_{j+1}^{(n)}, h(\hat{v}_{j+1}^{(n)}))\}_{n=1}^N; \theta_{\mathrm{ST}}\right).$$

4: Calculate the correction terms according to (4.28):

$$\left(\hat{w}_\theta^{(n)}, \hat{z}_\theta^{(n)}\right) = F^{\mathrm{gain}}\left(\hat{v}_{j+1}^{(n)}, h(\hat{v}_{j+1}^{(n)}), y_{j+1}^\dagger, f_v; \theta_{\mathrm{gain}}\right).$$

5: Calculate $(K_\theta^{(1)})_{j+1}$ and $(K_\theta^{(2)})_{j+1}$ (2.23).

6: Process with the neural network for localization:

$\hat{g}_{j+1} = F^{\mathrm{loc}}\left(f_v; \theta_{\mathrm{loc}}\right)$ according to (4.36).

7: Calculate localization matrices $(\lessdot^{(1)})_{j+1}$ and $(\lessdot^{(2)})_{j+1}$ (Subsection 4.2.3).

8: Calculate the Gain matrix $(K_\theta)_{j+1}$ with localization (4.34).

9: Get $\{v_{j+1}^{(n)}\}_{n=1}^N$ according to the analysis step:

$$v_{j+1}^{(n)} = \hat{v}_{j+1}^{(n)} + (K_\theta)_{j+1}\left(y_{j+1}^\dagger - \hat{y}_{j+1}^{(n)}\right)$$

10: Process with the neural network for inflation:

$\hat{u}_\theta^{(n)} = F^{\mathrm{infl}}(v_{j+1}^{(n)}, f_v; \theta_{\mathrm{infl}})$ according to (4.30).

11: Update the estimation with the learned inflation term $\hat{u}_\theta^{(n)}$ by

$$v_{j+1}^{(n)} \leftarrow v_{j+1}^{(n)} + \hat{u}_\theta^{(n)}.$$

12: **return** Ensemble $\{v_{j+1}^{(n)}\}_{n=1}^N$.

Algorithm 2. When we need to proceed with a different ensemble size $N' \neq N$, we follow Algorithm 3 to efficiently fine-tune part of the parameters.

After completing the pretraining (Algorithm 2) on ensemble size N, the inference procedure depends on the desired inference ensemble size N'. If $N' = N$, then no additional fine-tuning is required. However, if $N' \neq N$, for optimal performance, we recommend employing Algorithm 3 to efficiently fine-tune the model before inference. Below, we describe the inference steps for the general case where $N' \neq N$ after fine-tuning has been completed.

We first load the pre-trained parameter $\theta_{\mathrm{ST}}^{(N)}$ that remain unchanged regardless of ensemble size, along with the fine-tuned parameters $\theta_{\mathrm{gain}}^{(N')}$, $\theta_{\mathrm{infl}}^{(N')}$, and $\theta_{\mathrm{loc}}^{(N')}$ that were specifically optimized for ensemble size N'. We then initialize the ensemble $\{v_0^{(n)}\}_{n=1}^{N'}$ according to Eq. (4.37). For each time step j, we evolve the ensemble from $\{v_j^{(n)}\}_{n=1}^{N'}$ to $\{v_{j+1}^{(n)}\}_{n=1}^{N'}$ using Algorithm 1. This process can be extended to observation trajectories of arbitrary length. At each time step, we can approximate the true state $v_j^\dagger$ using the ensemble mean $\bar{v}_j = \frac{1}{N'} \sum_{n=1}^{N'} v_j^{(n)}$.

Algorithm 2 Pretraining.

Require: $J, M \in \mathbb{N}$; ensemble size N.
Ensure: Trained parameters $\theta^{(N)} = \{\theta_{\mathrm{ST}}^{(N)}, \theta_{\mathrm{gain}}^{(N)}, \theta_{\mathrm{infl}}^{(N)}, \theta_{\mathrm{loc}}^{(N)}\}$.

Training Data Generation: Generate M training trajectories with length $J+1$ and corresponding observations according to Subsection 4.3.
for each epoch, each minibatch indexed by $M_B \subset [M]$ **do**
  for $m \in M_B$ and the corresponding trajectory $\{v_j^\dagger\}_{j=1}^J$ **do**
    Initialize the ensemble $\{v_0^{(n)}\}_{n=1}^N$ (4.37).
    for $j = 0, 1, \ldots, J-1$ **do**
      Update from $\{v_j^{(n)}\}_{n=1}^N$ to $\{v_{j+1}^{(n)}\}_{n=1}^N$ according to Algorithm 1.
    end for
    Calculate the loss $\mathcal{L}_m$ (4.40).
  end for
  Calculate the batch loss $\mathcal{L}_{M_B}$ (4.41).
  Update θ via gradient descent on $\mathcal{L}_{M_B}$.
end for
return Optimized parameters $\theta^{(N)} = \{\theta_{\mathrm{ST}}^{(N)}, \theta_{\mathrm{gain}}^{(N)}, \theta_{\mathrm{infl}}^{(N)}, \theta_{\mathrm{loc}}^{(N)}\}$

Algorithm 3 Fine-tuning.

Require: Ensemble size $N' \neq N$; pretrained parameter $\theta_{\mathrm{ST}}^{(N)}$
Ensure: Updated parameters $\theta_{\mathrm{gain}}^{(N')}, \theta_{\mathrm{infl}}^{(N')}, \theta_{\mathrm{loc}}^{(N')}$ for new ensemble size N'

Use the same synthetic training data as Algorithm 2.
for each epoch, each minibatch indexed by $M_B \subset [M]$ **do**
  for $m \in M_B$ and the corresponding trajectory $\{v_j^\dagger\}_{j=1}^J$ **do**
    Initialize the ensemble $\{v_0^{(n)}\}_{n=1}^{N'}$ with size N'(4.37).
    for $j = 0, 1, \ldots, J-1$ **do**
      Update from $\{v_j^{(n)}\}_{n=1}^{N'}$ to $\{v_{j+1}^{(n)}\}_{n=1}^{N'}$ according to Algorithm 1.
    end for
    Calculate the loss $\mathcal{L}_m^{(N')}$ with ensemble size N' (4.40).
  end for
  Calculate the batch loss $\mathcal{L}_{M_B}^{(N')}$ (4.41) with ensemble size N'
  Update $\theta_{\mathrm{gain}}^{(N')}, \theta_{\mathrm{infl}}^{(N')}, \theta_{\mathrm{loc}}^{(N')}$ according to (4.44).
end for
return Optimized parameters $\theta_{\mathrm{gain}}^{(N')}, \theta_{\mathrm{infl}}^{(N')}, \theta_{\mathrm{loc}}^{(N')}$

B. Description of multihead attention

In Section 4.1, we introduce the definition of attention. However, in practical applications, multihead attention is commonly employed instead of the single-head variant. This appendix provides supplementary information on multihead attention, which is used in the set transformer architecture (Section 4.1) rather than standard single-head attention.

Recall the definition of the set of sequence of a finite length, $\mathcal{V}_F(\mathbb{R}^d) = \cup_{N=1}^\infty \mathcal{V}([N]; \mathbb{R}^d)$. In Section 4.1, we defined attention as a sequence-to-sequence operator $\mathsf{A} : \mathcal{V}_F(\mathbb{R}^{d_u}) \times \mathcal{V}_F(\mathbb{R}^{d_w}) \to \mathcal{V}_F(\mathbb{R}^{d_v})$ (4.3) with learnable parameters $\theta(\mathsf{A}) = \{Q, K, V\}$. For two

sequences $u \in \mathcal{U}([N];\mathbb{R}^{d_u})$ and $w \in \mathcal{U}([M];\mathbb{R}^{d_w})$, we have

$$\mathsf{A}(u,w)(j) = \frac{\sum_{j=1}^{M} \exp\big(\langle Qu(j), Kw(k)\rangle\big) V w(k)}{\sum_{\ell=1}^{M} \exp\big(\langle Qu(j), Kw(\ell)\rangle\big)}, \quad \forall j \in [N]. \tag{B.1}$$

In multihead attention, we employ several parallel attention heads, each with its own set of learnable parameters. This approach allows the model to jointly attend to information from different representation subspaces at different positions. Let us denote the multihead attention operator as $\mathsf{A}^R(\bullet,\bullet)$, where the superscript R indicates the number of attention heads. For each head $r \in [R]$, we have a separate set of learnable parameters $\theta(\mathsf{A}_r) = \{Q_r, K_r, V_r\}$, where $Q_r \in \mathbb{R}^{d_k \times d_u}$, $K_r \in \mathbb{R}^{d_k \times d_w}$, and $V_r \in \mathbb{R}^{\hat{d_V} \times d_w}$.

For each head r, we compute the attention output $\mathsf{A}_r(u,w)$ according to (B.1). The multihead attention mechanism $\mathsf{A}^R(u,w)$ combines the outputs from R heads through concatenation followed by linear projection. For $j \in [N]$, we compute the output as

$$\mathsf{A}^R(u,w)(j) = W^O\big[\mathsf{A}_1(u,w)(j); \mathsf{A}_2(u,w)(j); \dots; \mathsf{A}_R(u,w)(j)\big], \tag{B.2}$$

where $\big[\mathsf{A}_1(u,w)(j); \mathsf{A}_2(u,w)(j); \dots; \mathsf{A}_R(u,w)(j)\big] \in \mathbb{R}^{R\hat{d_V}}$ is a vector vertically concatenated from the outputs from all attention heads, and $W^O \in \mathbb{R}^{d_V \times (R\hat{d_V})}$ is an additional learnable parameter matrix that projects the concatenated outputs to the desired dimension.

The complete set of learnable parameters for the multihead attention mechanism is therefore:

$$\theta(\mathsf{A}^R) = \{Q_r, K_r, V_r\}_{r=1}^{R} \cup \{W^O\}. \tag{B.3}$$

In Section 3, we view attention as an operator on probability measures. This formulation naturally extends to multihead attention. The multihead attention operator acts on probability measures similarly to standard attention, but utilizes R parallel attention heads. In addition, we introduce two major properties of the set transformer architecture in Section 4.1, i.e. the permutation invariance and the adaptation to variable input lengths. These two properties are preserved for the version using multihead attention, since multihead attention with R heads effectively performs R standard attention operations in parallel and concatenates their results.

C. Implementation details

In this appendix, we provide additional implementation details to complement the methods presented in Section 4 and their application in the numerical experiments of Section 5.

The complete source code for reproducing our method is publicly available.[4] Additionally, for the benchmark methods, we performed extensive hyperparameter optimization using grid search, [5] implemented with the DAPPER package [58].

C.1. Feature dimensions

We provide detailed information about the latent dimensions used in our learning-based approach. While this information can be directly obtained from our code, we include this analysis to highlight two important properties of the set transformer architecture employed in our method: (1) the output is invariant to permutations of the input sequence, and (2) the output dimension remains consistent regardless of the length of the input sequence.

We introduce the general set transformer architecture in Section 4.1, where the encoder and decoder can include several self-attention blocks (SAB) $\mathsf{S}^{\mathrm{ST}}(\bullet)$ (4.18). In our experiments (Section 5), our actual architecture contains two SABs in both the encoder and decoder. This simplified structure provides sufficient representational capacity while maintaining computational efficiency. Our actual set tranformer architecture can be written as

$$\mathsf{F}^{\mathrm{ST}}(u) = \mathsf{F}_2^{\mathrm{NN}}(\bullet) \circ \mathsf{F}^{\mathrm{Cat}}(\bullet) \circ \mathsf{S}_4^{\mathrm{ST}}(\bullet) \circ \mathsf{S}_3^{\mathrm{ST}}(\bullet) \circ \mathsf{C}^{\mathrm{ST}}(s,\bullet) \circ \mathsf{S}_2^{\mathrm{ST}}(\bullet) \circ \mathsf{S}_1^{\mathrm{ST}}(\bullet) \circ \mathsf{F}_1^{\mathrm{NN}}(u), \tag{C.1}$$

where $\mathsf{F}_1^{\mathrm{NN}}$ and $\mathsf{F}_2^{\mathrm{NN}}$ are multilayer perceptrons consisting of multiple feedforward layers and activation layers (3.8); $\mathsf{S}_{1,2}^{\mathrm{ST}}$ and $\mathsf{S}_{3,4}^{\mathrm{ST}}$ are SABs serving as the encoder and decoder respectively; $\mathsf{F}^{\mathrm{Cat}}$ is a layer that concatenate all elements in a sequence into a long feature vector. Note that layers $\mathsf{S}_1^{\mathrm{ST}}$, $\mathsf{S}_2^{\mathrm{ST}}$, C^{ST}, $\mathsf{S}_3^{\mathrm{ST}}$ and $\mathsf{S}_4^{\mathrm{ST}}$ contain attention mechanisms, which are implemented as multihead attention with 8 heads (B).

Table C.1 provides the specific layer-wise dimension design in our architecture. We adopt similar network structures for all three experimental datasets: Lorenz '63 (L'63) (5.7), Lorenz '96 (L'96) (5.5), and Kuramoto-Sivashinsky (KS) (5.6). For all experiments, we use a trainable seed $s \in \mathcal{U}([16],\mathbb{R}^{64})$, which is a sequence of length 16 in $\mathbb{R}^{64}$. The first linear layer $\mathsf{F}_1^{\mathrm{NN}}$ maps input sequences of varying dimensions to $\mathbb{R}^{64}$, after which all subsequent layers maintain consistent dimensions across different dataset dynamics.

Table C.1 clearly illustrates the fact that the set transformer can process input sequences of arbitrary length and output fixed-dimension feature vectors. Specifically, regardless of the input sequence length N, after processing through the PMA block $\mathsf{C}^{\mathrm{ST}}(s,\bullet)$ (4.15), the sequence length always aligns with the seed length. This critical dimensional reduction occurs because the cross-attention mechanism in $\mathsf{C}^{\mathrm{ST}}(s,\bullet)$ uses the fixed-length seed s as queries to attend to the input sequence. The output dimension is then determined by the concluding $\mathsf{F}_2^{\mathrm{NN}}$ layer. For detailed computational mechanisms, please refer to Sections 3 and 4.1.

The table also helps illustrate where the permutation invariance is established in the architecture. Before the PMA $\mathsf{C}^{\mathrm{ST}}(s,\bullet)$, the latent features are permutation equivariant with respect to the input, meaning that if the input sequence elements are reordered,

[4] https://github.com/wispcarey/DALearning.

[5] https://github.com/wispcarey/DapperGridSearch.

the corresponding features will be reordered in the same way. However, after the PMA block, the feature ordering becomes entirely dependent on the seed sequence and completely independent of the input sequence ordering. This transformation from permutation equivariance to permutation invariance is a crucial property (Proposition 2) of the set transformer, enabling them to process sets rather than sequences.

C.2. An example of learning the localization weight matrix

Here we provide a concrete example of how to compute the parameterized localization weight matrices $\lessdot^{(1)}$ and $\lessdot^{(2)}$ in the Lorenz '96 system (Section 5.2) with state dimension $d_v = 40$ and observation dimension $d_y = 10$. Specifically, the 40 state variables are indexed by $1, 2, \dots, 40$ in a cyclic manner, and the observations are available at indices $4, 8, 12, \dots, 40$ (i.e., every 4th state variable). We employ a periodic distance metric for all index pairs $k, \ell \in [d_v]$,

$$d_{\text{peri}}(k, \ell) = \min\big(|k - \ell|, 40 - |k - \ell|\big). \tag{C.2}$$

Then we can consider all the unique distance values given by:

$$\{D_{k\ell} = d_{\text{peri}}(k, \ell) | k, \ell \in [40]\} = \{0, 1, \dots, 20\}. \tag{C.3}$$

Recall that $\circ$ denotes the Hadamard pointwise matrix product. In the classic localization setting (4.33) for the EnKF approach,

$$K = \hat{C}^{vh} \circ L^{vh} \big(\hat{C}^{hh} \circ L^{hh} + \Gamma\big)^{-1}, \tag{C.4}$$

we have

$$\begin{aligned} (L^{vh})_{k,\ell} &= g(d_{\text{peri}}(k, 4\ell)), \quad k = 1, 2, \dots, 40, \quad \ell = 1, 2, \dots, 10, \\ (L^{hh})_{k,\ell} &= g(d_{\text{peri}}(4k, 4\ell)), \quad k = 1, 2, \dots, 10, \quad \ell = 1, 2, \dots, 10. \end{aligned} \tag{C.5}$$

In our learning framework introduced in Section 4.2.3, we learn a vector $\hat{g} \in \mathbb{R}^{21}$ according to (4.36) for the function values $g_\theta(D)$ for $D = 0, 1, 2, \dots, 20$. Then in our proposed scheme (4.34),

$$K_\theta = K_\theta^{(1)} \circ \lessdot^{(1)} \left(K_\theta^{(2)} \circ \lessdot^{(2)} + \Gamma\right)^{-1}, \tag{C.6}$$

we can calculate the parameterized localization weight matrices by

$$\begin{aligned} (\lessdot^{(1)})_{k,\ell} &= g_\theta(d_{\text{peri}}(k, 4\ell)), \quad k = 1, 2, \dots, 40, \quad \ell = 1, 2, \dots, 10, \\ (\lessdot^{(2)})_{k,\ell} &= g_\theta(d_{\text{peri}}(4k, 4\ell)), \quad k = 1, 2, \dots, 10, \quad \ell = 1, 2, \dots, 10. \end{aligned} \tag{C.7}$$

C.3. Initial conditions

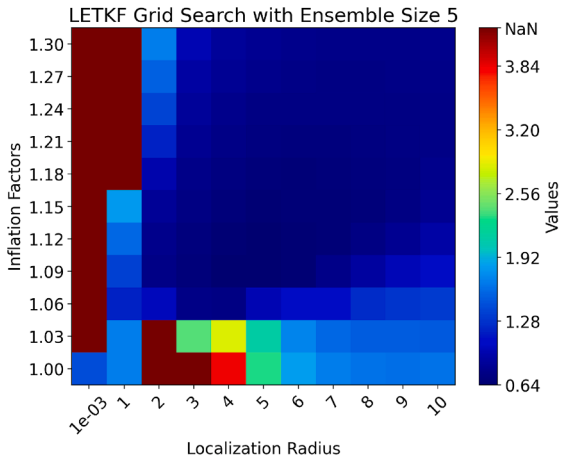
Here we provide details of the initial conditions for the experiments in Section 5. To ensure our initial states lie on the attractor, we first select a base value and then evolve the system's dynamics forward for a substantial number of time steps. This serves as a "burn-in" period. The initial conditions detailed below for each system are subsequently used to generate all trajectories for pretraining, fine-tuning, and testing.

1. **Lorenz '96** (Section 5.2): Sample $x_0 \sim \mathsf{N}(5, I_{40})$ and run the forward dynamic (5.5) for random 10^3 to 5×10^5 time steps with step size $\Delta t = 0.15$ to get $v_0^\dagger$. Within each Δt time interval, there are 5 RK4 numerical integration steps with the step size $\Delta t/5 = 0.03$. The initial distribution is set to be $\mathsf{N}(v_0^\dagger, I_{40})$.
2. **Kuramoto–Sivashinsky** (Section 5.3): Set the interval $[0, L]$ as $L = 32\pi$. For $u_0(x) = \cos(2x/L)(1 + \sin(2x/L))$, run the forward dynamic (5.6) for random 10^3 to 5×10^5 time steps with step size $\Delta t = 1.00$ to get $\hat{u}$. Within each Δt time interval, there are 4 RK4 numerical integration steps with the step size $\Delta t/4 = 0.25$. The initial state $v_0^\dagger = (\hat{u}(x_0), \dots, \hat{u}(x_{127}))$ with $x_\ell = \ell L/128$, $\ell = 0, 1, \dots, 127$. The initial distribution is set to be $\mathsf{N}(v_0^\dagger, I_{128})$.
3. **Lorenz '63** (Section 5.4): Sample $x_0 \sim \mathsf{N}(0, I_3)$ and run the forward dynamic (5.7) for random 10^3 to 5×10^5 time steps with step size $\Delta t = 0.15$ to get $v_0^\dagger$. Within each Δt time interval, there are 5 RK4 numerical integration steps with the step size $\Delta t/5 = 0.03$. The initial distribution is set to be $\mathsf{N}(v_0^\dagger, I_3)$.

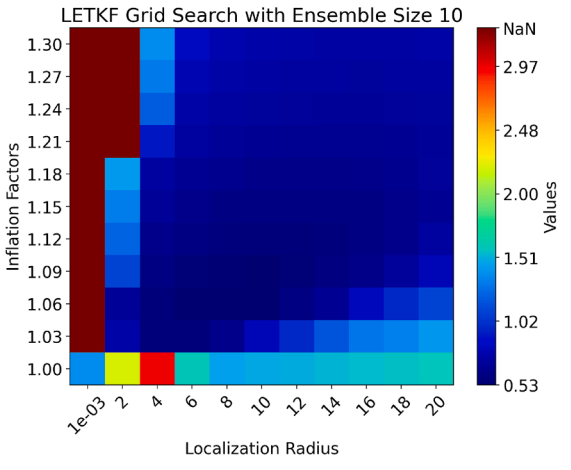
C.4. Hyperparameter setting for our training

We provide details of the training hyperparameters in experiments. The parameters listed in Table C.2 are utilized for pretraining with an ensemble size of $N = 10$ in the experiments detailed in Section 5. When fine-tuning with a different ensemble size $N' \neq N$ (Section 4.4), the number of training trajectories is halved compared to the quantity used for pretraining. Furthermore, the learning rate for fine-tuning is set to $1/10$ of the learning rate used during pretraining. We use the AdamW optimizer [65] for both the pretraining and fine-tuning.

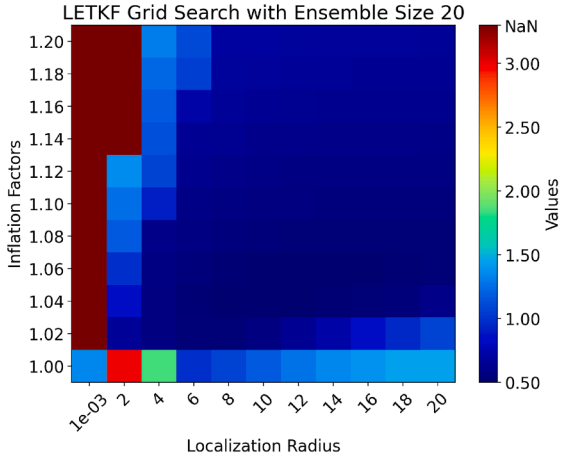
During the training process, we implement a clamping mechanism to ensure numerical stability and prevent issues such as exploding estimated trajectories. If the absolute value of any dimension of an estimated particle exceeds the specified clamp value (provided in the "Clamp" column of Table C.2), its sign is preserved, but its absolute value is replaced by the clamp threshold.



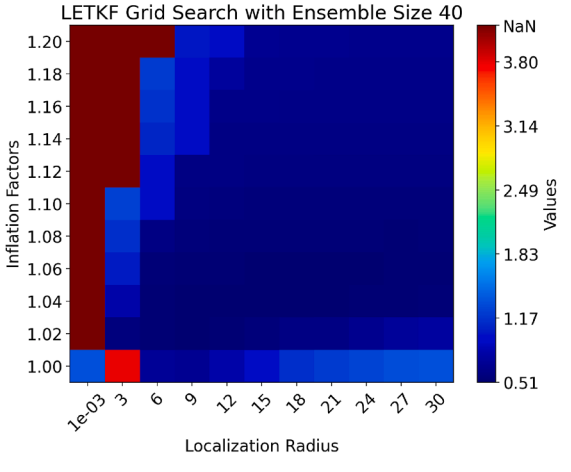
(a) $N = 5$



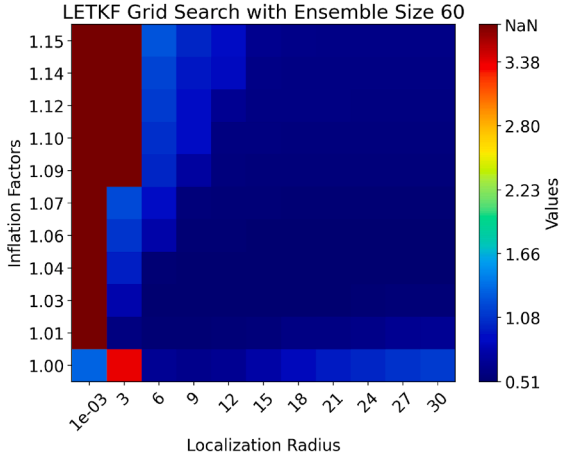
(b) $N = 10$



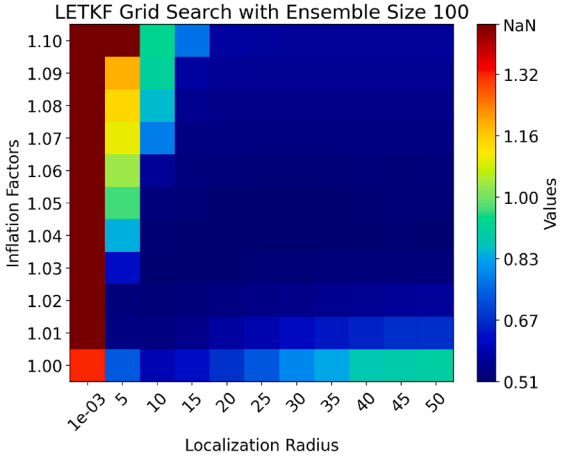
(c) $N = 20$



(d) $N = 40$



(e) $N = 60$



(f) $N = 100$

Fig. C.1. Grid search results for the Local Ensemble Transform Kalman Filter (LETKF) [15] on the Kuramoto–Sivashinsky (KS) dynamical system (5.6) varying ensemble sizes. Each subplot shows the RMSE performance across different combinations of inflation parameter α and localization radius parameter r. Darker colors indicate lower relative RMSE (R-RMSE) values (5.3) (better performance).

Table C.1
Feature dimensions at each layer of the set transformer architecture to process the ensemble states for different dynamical systems, Lorenz '63 (L'63) (5.7), Lorenz '96 (L'96) (5.5), and Kuramoto-Sivashinsky (KS) (5.6). We use (N, d) to denote a sequence in $\mathbb{R}^d$ with length N (i.e. in $\mathcal{U}([N], \mathbb{R}^d)$). The trainable seed is a sequence in $\mathbb{R}^{64}$ with length 16. The input sequence dimension is the sum of state dimension and the observation dimension.

Dataset	Feature Dimensions after Each Layer								
	Input	F_1^{NN}	S_1^{ST}	S_2^{ST}	$C^{ST}(s, \bullet)$	S_3^{ST}	S_4^{ST}	F^{Cat}	F_2^{NN}
L'63	(N,4)	(N,64)	(N,64)	(N,64)	(16,64)	(16,64)	(16,64)	1024	64
L'96	(N,50)								
KS	(N,144)								

Table C.2
Training hyperparameters for pretraining with an ensemble size N = 10. For fine-tuning with N' ≠ N, the number of training trajectories is halved, and the learning rate is reduced to 1/10 of the values listed. Obs Δt refers to the observation time step.

Dataset	Train Traj		Test Traj		Obs	Learning	Batch	Clamp
	Num	Length	Num	Length	Δt	Rate	Size	
Lorenz '96	8192	60	64	1500	0.15	1e-3	512	20
KS	8192	60	64	2000	1.00	5e-4	256	10
Lorenz '63	8192	60	64	1500	0.15	1e-3	1024	60

C.5. Hyperparameter optimization via grid search

In our experiments (Section 5), we compare our pretrained and fine-tuned methods introduced in Section 4 with benchmarks, EnKF [9], ESRF [11], LETKF [15], and iEnKF[16], detailed in Section 5.1. The benchmark methods we compared require hyperparameter tuning for different ensemble sizes so that they are optimized at each ensemble size and hence indeed present a challenging benchmark for our proposed methodology. We use the DAPPER package [58], which implements these benchmark methods, allowing us to focus solely on conducting grid searches for the required hyperparameters across various ensemble sizes. This appendix presents examples of our grid search results to provide insight into the comprehensive range of our hyperparameter optimization. Complete results can be found in our GitHub repository.[6]

For each of the four benchmark methods mentioned above, we perform grid search on the inflation parameter $\alpha > 1$. After the analysis step, the state $v^{(n)}$ is updated according to:

$$v^{(n)} \leftarrow v^{(n)} + (\alpha - 1)\left(v^{(n)} - \frac{1}{N}\sum_{\ell=1}^{N} v^{(\ell)}\right), \quad n = 1, 2, \ldots, N. \tag{C.8}$$

In addition to inflation, for LETKF specifically, we consider the localization radius parameter r, which is used in the Gaspari–Cohn (GC) function [53] to map distances to localization weights.

For different methods, datasets, and ensemble sizes, we employ an adaptive approach to determine the grid search step sizes and ranges. Our evaluation metric is the average relative RMSE (R-RMSE) (5.3) computed over 64 test trajectories. Lower values of this metric indicate superior performance. Initially, we establish a consistent range and step size across all scenarios, then refine these parameters based on preliminary performance results in each specific case. This adaptive refinement allows us to concentrate computational resources on the most promising hyperparameter regions.

Fig. C.1 illustrates our grid search results for the Local Ensemble Transform Kalman Filter (LETKF) [15] method applied to the Kuramoto–Sivashinsky (KS) dynamical system (5.6) across different ensemble sizes ($N = 5, 10, 20, 40, 60, 100$). Each heat map displays the R-RMSE performance for various combinations of inflation parameter α and localization radius r.

C.6. Additional experiments: Linear setting

Our MNMEF aims to learn correction terms based on the EnKF scheme; see the mean-field version (2.21)–(2.24) and their particle counterparts (4.1), (4.27). The motivation behind MNMEF is that EnKF is not an exact scheme for nonlinear or non-Gaussian settings from a mean-field perspective, so learnable corrections may close the modeling gap. When the forward dynamics and observation models are linear,

$$\Psi(v) = Av, \qquad h(v) = Hv, \tag{C.9}$$

[6] https://github.com/wispcarey/DapperGridSearch.

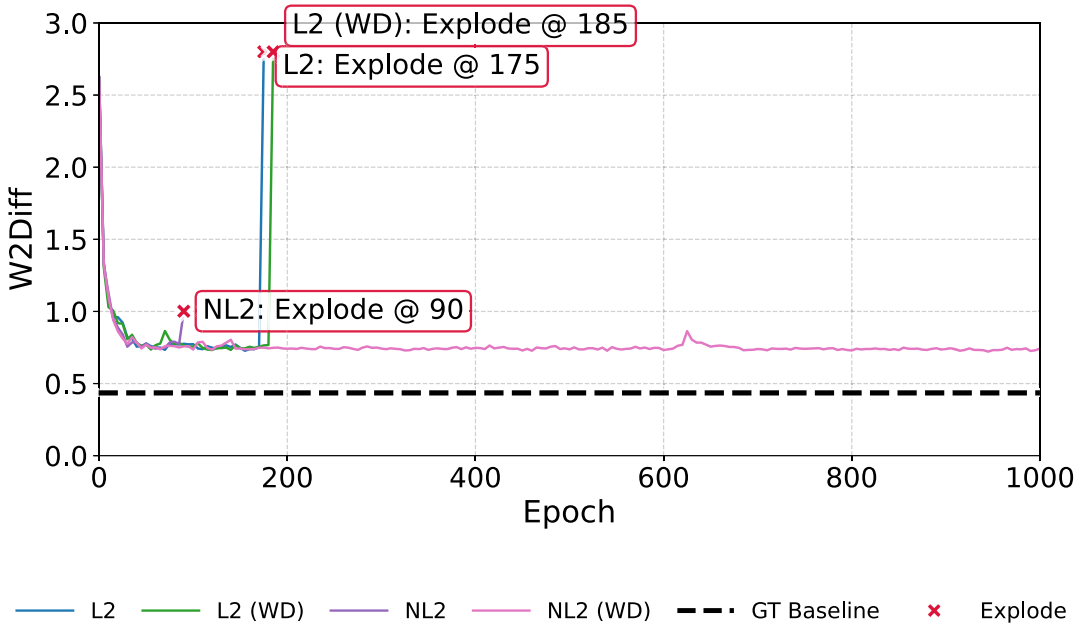


Fig. C.2. Linear–Gaussian experiment. Test W_2 (ensemble Gaussian vs. Kalman Gaussian) versus training epoch under the four settings in (C.13), evaluated on 64 held-out trajectories (length 500 each). *Legend (from the lower panel):* L2 = unnormalized loss (C.12); NL2 = normalized loss (C.11); WD = weight decay 10^{-2}; GT Baseline = finite-ensemble sampling baseline obtained by drawing ensembles directly from the Kalman Gaussian and computing W_2 via (C.14). Explode marks the divergence for each curve (happening within 5 epochs of the mark). Consistent with the text, the NL2 (WD) configuration remains stable for 1000 epochs and approaches the baseline, whereas other variants are unstable.

with additive Gaussian noises as in (2.1)–(2.2), the Kalman filter is exact, and the EnKF recovers it in the mean-field/infinite-ensemble limit. In this setting, the theoretically correct choice for the correction terms is

$$\hat{w}_\theta \equiv 0, \qquad \hat{z}_\theta \equiv 0, \tag{C.10}$$

so that K_θ reduces to the classical Kalman gain (2.11b). Any nonzero learned corrections cannot improve optimality and therefore risk modeling observation/forecast noise rather than signal. Because the linear–Gaussian case is already optimal, training MNMEF with freely learnable $(\hat{w}_\theta, \hat{z}_\theta)$ can overfit stochastic fluctuations: with sufficiently many epochs (and without explicit regularization/early stopping), the network may drive the corrections to fit noise realizations. Although $K_\theta^{(2)} + \Gamma$ remains invertible (by positive definiteness of Γ), such noise-fitting can manifest as unstable updates and lead to filter divergence. We provide a minimal working example in our repository.[7]

To make the discussion concrete, we run a controlled linear–Gaussian experiment. The state dimension is $d_v = 10$ with a matrix $A \in \mathbb{R}^{10\times 10}$ whose eigenvalues satisfy $|\lambda_i(A)| = 1$ for all i; this yields marginally stable linear dynamics. The observation dimension is $d_y = 5$ with $H \in \mathbb{R}^{5\times 10}$ that observes every other coordinate. The process noise covariance is $\Sigma = \sigma_v^2 I$ with $\sigma_v = 0.01$, the observation noise covariance is $\Gamma = \sigma_y^2 I$ with $\sigma_y = 1$, and the initial condition is $v_0 \sim \mathcal{N}(0, I)$.

For training, the default loss of our MNMEF is the normalized loss used in the main text (4.40) and repeated here for completeness,

$$\mathcal{L}_m(\theta) = \frac{1}{J}\sum_{j=1}^{J} \frac{\|\bar{v}_j(\theta) - v_j^\dagger\|_2^2}{\|v_j^\dagger\|_2^2}, \tag{C.11}$$

which enforces scale invariance across trajectories. We also consider the unnormalized alternative

$$\widetilde{\mathcal{L}}_m(\theta) = \frac{1}{J}\sum_{j=1}^{J} \left\|\bar{v}_j(\theta) - v_j^\dagger\right\|_2^2, \tag{C.12}$$

[7] https://github.com/wispcarey/DALearning./

and, orthogonally, we optionally add weight decay with coefficient 10^{-2}, which is exactly equivalent to adding an ℓ_2 penalty to the training objective. Combining these choices yields four settings:

$$\mathcal{L}_{\text{train}}(\theta) = \begin{cases} \mathcal{L}_m(\theta) + 10^{-2}\,\|\theta\|_2^2 & \text{(NL2 (WD): normalized loss + weight decay)} \\ \mathcal{L}_m(\theta) & \text{(NL2: normalized loss, no weight decay)} \\ \widetilde{\mathcal{L}}_m(\theta) + 10^{-2}\,\|\theta\|_2^2 & \text{(L2 (WD): unnormalized loss + weight decay)} \\ \widetilde{\mathcal{L}}_m(\theta) & \text{(L2: unnormalized loss, no weight decay).} \end{cases} \tag{C.13}$$

We train with learning rate 10^{-3} for 1000 epochs on 8192 trajectories, each of length 60. Because the model is linear–Gaussian, the exact filtering distribution at every time step is Gaussian and can be computed by the Kalman filter, providing ground-truth means and covariances against which to evaluate.

As the metric, we compare at each step the ensemble-based Gaussian approximation (empirical mean and covariance of the ensemble) with the Kalman ground-truth Gaussian via the 2-Wasserstein distance. For $\mathcal{N}(m_1, C_1)$ and $\mathcal{N}(m_2, C_2)$, the closed-form expression is

$$W_2^2\big(\mathcal{N}(m_1, C_1),\, \mathcal{N}(m_2, C_2)\big) = \|m_1 - m_2\|_2^2 \;+\; \mathrm{Tr}\Big(C_1 + C_2 - 2\big(C_2^{1/2} C_1 C_2^{1/2}\big)^{1/2}\Big). \tag{C.14}$$

We report the test W_2 on 64 held-out trajectories (each of length 500 and disjoint from the training set), averaged over time and across trajectories, as a function of training epoch.

A finite-ensemble effect sets an irreducible baseline: even if an ensemble is drawn exactly from the true Kalman Gaussian at each step, the empirical mean and covariance estimated from a finite ensemble will not match the truth perfectly, hence the W_2 discrepancy cannot be zero. We therefore include a finite-ensemble sampling baseline by i.i.d. sampling from the Kalman Gaussian with the same ensemble size and computing (C.14); this quantifies the best attainable error due purely to sampling.

The results are summarized in Fig. C.2. Training with the normalized loss (C.11) without weight decay eventually exhibits exploding W_2, consistent with the correction pathway overfitting observation/process noise when no true signal remains to learn. Adding weight decay (normalized + WD) stabilizes training throughout 1000 epochs and closely tracks the finite-ensemble baseline. In contrast, the unnormalized loss (C.12) leads to instability regardless of weight decay: both unnormalized variants eventually diverge. This confirms that, in the strictly linear–Gaussian regime where the Kalman filter is optimal and the correct corrections are $(\hat{w}_\theta, \hat{z}_\theta) \equiv (0, 0)$, normalization and explicit ℓ_2 regularization are critical to avoid learning spurious noise-driven updates.

By contrast, on nonlinear systems (Lorenz '96, Kuramoto–Sivashinsky, and Lorenz '63), where the Gaussian ansatz underlying EnKF is only approximate, the correction terms provide genuine modeling capacity. In those settings, even with very long training we did not observe noise overfitting or filter divergence; the learned corrections consistently improved or matched EnKF performance across our runs. In summary, in linear–Gaussian settings practitioners should disable the correction pathway or regularize it to recover the Kalman solution, whereas in nonlinear settings allowing trainable corrections is beneficial and did not cause instability in our experiments.

References

[1] A.H. Jazwinski, Stochastic Processes and Filtering Theory, 64 of *Mathematics in Science and Engineering*, Academic Press, New York, New York, 1970.

[2] K. Law, A. Stuart, K. Zygalakis, Data Assimilation: A Mathematical Introduction, 62 of *Texts in Applied Mathematics*, Springer, Cham, Cham, 2015. https://doi.org/10.1007/978-3-319-20325-6

[3] M. Asch, M. Bocquet, M. Nodet, Data Assimilation: Methods, Algorithms, and Applications, 11 of *Fundamentals of Algorithms*, SIAM, Philadelphia, Philadelphia, 2016. https://doi.org/10.1137/1.9781611974546

[4] S. Reich, C. Cotter, Probabilistic Forecasting and Bayesian Data Assimilation, Cambridge University Press, Cambridge, Cambridge, 2015. https://doi.org/10.1017/CBO9781107706804

[5] G. Evensen, F.C. Vossepoel, P.J. van Leeuwen, Data Assimilation Fundamentals: A Unified Formulation of the State and Parameter Estimation Problem, Springer Textbooks in Earth Sciences, Geography and Environment, Springer, Cham, Cham, 2022. https://doi.org/10.1007/978-3-030-96709-3

[6] E. Bach, R. Baptista, D. Sanz-Alonso, A. Stuart, Machine Learning for Inverse Problems and Data Assimilation, 2025.

[7] R.E. Kalman, A new approach to linear filtering and prediction problems, J. Basic Eng. 82 (1) (1960) 35–45. https://doi.org/10.1115/1.3662552

[8] G. Evensen, The ensemble Kalman filter: theoretical formulation and practical implementation, Ocean Dyn. 53 (2003) 343–367.

[9] G. Burgers, P.J. Van Leeuwen, G. Evensen, Analysis scheme in the ensemble Kalman filter, Mon. Weather Rev. 126 (6) (1998) 1719–1724.

[10] J.L. Anderson, An ensemble adjustment Kalman filter for data assimilation, Mon. Weather Rev. 129 (12) (2001) 2884–2903. https://doi.org/10.1175/1520-0493(2001)129<2884:AEAKFF>2.0.CO;2

[11] M.K. Tippett, J.L. Anderson, C.H. Bishop, T.M. Hamill, J.S. Whitaker, Ensemble square root filters, Mon. Weather Rev. 131 (7) (2003) 1485–1490. Publisher: American Meteorological Society Section: Monthly Weather Review. https://doi.org/10.1175/1520-0493(2003)131<1485:ESRF>2.0.CO;2

[12] P.J. Van Leeuwen, A consistent interpretation of the stochastic version of the ensemble Kalman filter, Q. J. R. Meteorolog. Soc. 146 (731) (2020) 2815–2825.

[13] J.L. Anderson, S.L. Anderson, A Monte Carlo implementation of the nonlinear filtering problem to produce ensemble assimilations and forecasts, Mon. Weather Rev. 127 (12) (1999) 2741–2758. https://doi.org/10.1175/1520-0493(1999)127<2741:AMCIOT>2.0.CO;2

[14] J.L. Anderson, An adaptive covariance inflation error correction algorithm for ensemble filters, Tellus A Dyn. Meteorol. Oceanogr. 59 (2) (2007) 210–224. https://doi.org/10.1111/j.1600-0870.2006.00216.x

[15] B.R. Hunt, E.J. Kostelich, I. Szunyogh, Efficient data assimilation for spatiotemporal chaos: a local ensemble transform Kalman filter, Phys. D 230 (1) (2007) 112–126.

[16] P. Sakov, D.S. Oliver, L. Bertino, An iterative EnKF for strongly nonlinear systems, Mon. Weather Rev. 140 (6) (2012) 1988–2004.

[17] M. Zupanski, Maximum likelihood ensemble filter: theoretical aspects, Mon. Weather Rev. 133 (6) (2005) 1710–1726.

[18] S. Cheng, C. Quilodrán-Casas, S. Ouala, A. Farchi, C. Liu, P. Tandeo, R. Fablet, D. Lucor, B. Iooss, J. Brajard, D. Xiao, T. Janjic, W. Ding, Y. Guo, A. Carrassi, M. Bocquet, R. Arcucci, Machine learning with data assimilation and uncertainty quantification for dynamical systems: a review, IEEE/CAA J. Autom. Sin. 10 (6) (2023) 1361–1387. https://doi.org/10.1109/JAS.2023.123537

[19] H.S. Hoang, P. De Mey, O. Talagrand, A simple adaptive algorithm of stochastic approximation type for system parameter and state estimation, in: Proceedings of 1994 33rd IEEE Conference on Decision and Control, 1, 1994, pp. 747–752 vol.1. https://doi.org/10.1109/CDC.1994.410863

[20] N. Mallia-Parfitt, J. Bröcker, Assessing the performance of data assimilation algorithms which employ linear error feedback, Chaos Interdiscip. J. Nonlinear Sci. 26 (10) (2016) 103109. https://doi.org/10.1063/1.4965029
[21] M. Levine, A. Stuart, A framework for machine learning of model error in dynamical systems, Commun. Am. Math. Soc. 2 (07) (2022) 283–344. https://doi.org/10.1090/cams/10
[22] E. Bach, R. Baptista, E. Luk, A. Stuart, Learning optimal filters using variational inference, 2025, https://doi.org/10.48550/arXiv.2406.18066
[23] M. McCabe, J. Brown, Learning to assimilate in chaotic dynamical systems, in: Advances in Neural Information Processing Systems, 34, Curran Associates, Inc., 2021, pp. 12237–12250.
[24] M. Bocquet, A. Farchi, T.S. Finn, C. Durand, S. Cheng, Y. Chen, I. Pasmans, A. Carrassi, Accurate deep learning-based filtering for chaotic dynamics by identifying instabilities without an ensemble, Chaos Interdiscip. J. Nonlinear Sci. 34 (9) (2024) 091104. https://doi.org/10.1063/5.0230837
[25] A. Spantini, R. Baptista, Y. Marzouk, Coupling techniques for nonlinear ensemble filtering, SIAM Rev. 64 (4) (2022) 921–953.
[26] H.G. Chipilski, Exact nonlinear state estimation, J. Atmos. Sci. 82 (4) (2025) 809–827. Publisher: American Meteorological Society. Boston, MA, USA, https://journals.ametsoc.org/view/journals/atsc/82/4/JAS-D-24-0171.1.xml, https://doi.org/10.1175/JAS-D-24-0171.1
[27] Y. Shi, V. De Bortoli, G. Deligiannidis, A. Doucet, Conditional simulation using diffusion Schrödinger bridges, in: Uncertainty in Artificial Intelligence, PMLR, 2022, pp. 1792–1802.
[28] M. Al-Jarrah, N. Jin, B. Hosseini, A. Taghvaei, Nonlinear filtering with Brenier optimal transport maps, arXiv preprint arXiv:2310.13886 (2023).
[29] P.J. Van Leeuwen, H.R. Künsch, L. Nerger, R. Potthast, S. Reich, Particle filters for high-dimensional geoscience applications: a review, Q. J. R. Meteorolog. Soc. 145 (723) (2019) 2335–2365.
[30] C.-C. Hu, P.J. Van Leeuwen, J.L. Anderson, An implementation of the particle flow filter in an atmospheric model, Mon. Weather Rev. 152 (10) (2024) 2247–2264.
[31] F. Daum, J. Huang, Particle flow for nonlinear filters, in: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2011, pp. 5920–5923.
[32] K. Bergemann, S. Reich, An ensemble Kalman-Bucy filter for continuous data assimilation, Meteorol. Z. 21 (3) (2012) 213.
[33] M. Pulido, P.J. van Leeuwen, Kernel embedding of maps for Bayesian inference: the variational mapping particle filter, in: EGU General Assembly Conference Abstracts, 2018, p. 3750.
[34] J. Bröcker, D. Engster, U. Parlitz, Probabilistic evaluation of time series models: a comparison of several approaches, Chaos Interdiscip. J. Nonlinear Sci. 19 (4) (2009) 043130. https://doi.org/10.1063/1.3271343
[35] P. Boudier, A. Fillion, S. Gratton, S. Gürol, S. Zhang, Data assimilation networks, J. Adv. Model. Earth Syst. 15 (4) (2023) e2022MS003353. https://doi.org/10.1029/2022MS003353
[36] X.-H. Zhou, Z.-R. Liu, H. Xiao, BI-EqNO: generalized approximate Bayesian inference with an equivariant neural operator framework, 2024, https://doi.org/10.48550/arXiv.2410.16420
[37] K. Höhlein, B. Schulz, R. Westermann, S. Lerch, Postprocessing of ensemble weather forecasts using permutation-invariant neural networks, Artif. Intell. Earth Syst. 3 (1) (2024) e230070. arXiv:2309.04452 [physics, stat]. https://doi.org/10.1175/AIES-D-23-0070.1
[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Advances in Neural Information Processing Systems, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, (Eds.), Attention is All you Need, 30, Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
[39] S. Cao, Choose a transformer: fourier or galerkin, in: M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, J.W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 34, Curran Associates, Inc., 2021, pp. 24924–24940.
[40] Z. Li, K. Meidani, A.B. Farimani, Transformer for partial differential equations' operator learning, Trans. Mach. Learn. Res. (2023) 1–34. https://openreview.net/forum?id=EPPqt3uERT.
[41] E. Calvello, N.B. Kovachki, M.E. Levine, A.M. Stuart, Continuum attention for neural operators, arXiv preprint arXiv:2406.06486 (2024).
[42] S. Wang, J.H. Seidman, S. Sankaran, H. Wang, G.J. Pappas, P. Perdikaris, CViT: continuous vision transformer for operator learning, in: The Thirteenth International Conference on Learning Representations, 2025.
[43] B. Geshkovski, C. Letrouit, Y. Polyanskiy, P. Rigollet, A mathematical perspective on transformers, arXiv preprint arXiv:22312.10794 (2024).
[44] V. Castin, P. Ablin, J.A. Carrillo, G. Peyré, A unified perspective on the dynamics of deep transformers, arXiv preprint arXiv:2501.18322 (2025).
[45] B. Geshkovski, P. Rigollet, D. Ruiz-Balet, Measure-to-measure interpolation using Transformers, arXiv preprint arXiv:2411.04551 (2024).
[46] A. Bain, D. Crisan, Fundamentals of Stochastic Filtering, 3, Springer, 2009.
[47] A.M. Stuart, Inverse problems: a Bayesian perspective, Acta. Numer. 19 (2010) 451–559.
[48] E. Calvello, S. Reich, A.M. Stuart, Ensemble Kalman methods: a mean field perspective, Acta Numerica, arXiv preprint arXiv:2209.11371 (2025).
[49] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: learning maps between function spaces with applications to PDEs, J. Mach. Learn. Res. 24 (89) (2023) 1–97.
[50] N.B. Kovachki, S. Lanthaler, A.M. Stuart, Operator learning: algorithms and analysis (2024), arXiv preprint arXiv:2402.15715.
[51] E. Calvello, P. Monmarché, A.M. Stuart, U. Vaes, Accuracy of the ensemble Kalman filter in the near-linear setting (2024), arXiv preprint arXiv:2409.09800.
[52] J. Lee, Y. Lee, J. Kim, A.R. Kosiorek, S. Choi, Y.W. Teh, Set transformer: a framework for attention-based permutation-invariant neural networks (2019), arXiv preprint arXiv:1810.00825.
[53] G. Gaspari, S.E. Cohn, Construction of correlation functions in two and three dimensions, Q. J. R. Meteorolog. Soc. 125 (554) (1999) 723–757. https://doi.org/10.1002/qj.49712555417
[54] D. Vishny, M. Morzfeld, K. Gwirtz, E. Bach, O.R.A. Dunbar, D. Hodyss, High-dimensional covariance estimation from a small number of samples, J. Adv. Model. Earth Syst. 16 (9) (2024) e2024MS004417. https://doi.org/10.1029/2024MS004417
[55] W. Sacher, P. Bartello, Sampling errors in ensemble kalman filtering. part i: theory, Mon. Weather Rev. 136 (8) (2008) 3035–3049. https://doi.org/10.1175/2007MWR2323.1
[56] P. Sakov, P.R. Oke, A deterministic formulation of the ensemble Kalman filter: an alternative to ensemble square root filters, Tellus A Dyn. Meteorol. Oceanogr. 60 (2) (2008) 361–371.
[57] C.H. Bishop, B.J. Etherton, S.J. Majumdar, Adaptive sampling with the ensemble transform Kalman filter. part I: theoretical aspects, Mon. Weather Rev. 129 (3) (2001) 420–436.
[58] P.N. Raanes, Y. Chen, C. Grudzien, et al., DAPPER: data assimilation with python: a package for experimental research, J. Open Source Software 9 (94) (2024) 5150. https://doi.org/10.21105/joss.05150
[59] J. Stoer, R. Bulirsch, R. Bartels, W. Gautschi, C. Witzgall, Introduction to Numerical Analysis, 1993, Springer, 1980.
[60] E.N. Lorenz, Predictability: a problem partly solved, in: Proc. Seminar on Predictability, 1, Reading, 1996, pp. 1–18.
[61] Y. Kuramoto, Diffusion-induced chaos in reaction systems, Prog. Theor. Phys. Suppl. 64 (1978) 346–367.
[62] D.M. Michelson, G.I. Sivashinsky, Nonlinear analysis of hydrodynamic instability in laminar flames–II. numerical experiments, Acta Astronaut. 4 (11–12) (1977) 1207–1221.
[63] A.-K. Kassam, L.N. Trefethen, Fourth-order time-stepping for stiff PDEs, SIAM J. Sci. Comput. 26 (4) (2005) 1214–1233.
[64] E.N. Lorenz, Deterministic nonperiodic flow, J. Atmos. Sci. 20 (2) (1963) 130–148. https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2
[65] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).