A PRETRAINED MODEL FOR BAYESIAN FILTERING IN HETEROGENEOUS DYNAMICAL SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Filtering has not yet had its foundation-model moment: state estimation is still dominated either by online Bayesian filtering algorithms that require known dynamics as inputs, or by learning-based neural filters trained from scratch for a single system or parametric family. The core challenge is structural heterogeneity: dynamical systems differ in their governing dynamics, physical scales, and noise levels, precluding a shared representation space for a reusable filter. To bridge this gap, we introduce *FilterFM*, the *first* pretrained model for Bayesian filtering that estimates latent states across heterogeneous dynamical systems with a single checkpoint. FilterFM employs a hierarchical two-level observation sequence encoder and a prior-to-posterior state-reconstruction module that together handle varying observation and state dimensions across heterogeneous systems, while modeling state uncertainty via quantile prediction. Trained on a mixture of heterogeneous filtering tasks, FilterFM exhibits superior in-domain performance, delivering significant gains in estimation accuracy, runtime efficiency, and noise robustness compared to existing baselines. Beyond in-domain performance, it marks the first model to successfully enable zero-shot filtering on unseen dynamics, bridging the generalization gap in state estimation. Furthermore, we uncover clear scaling behaviors in foundation-model-based Bayesian filtering, which combined with comprehensive component ablation studies, validate the structural optimality of our design and broaden the generalization scope for this research field.

1 INTRODUCTION

Bayesian filtering serves as a basic tool for state estimation in robotics (Barfoot, 2024), navigation (Ko & Fox, 2009), atmospheric science (Hickman et al., 2025), chemical process monitoring (Wang & Chiang, 2019), and many other scientific domains (Zhang et al., 2024; Course & Nair, 2023). Although foundation models have reshaped multi-modal modeling and time-series forecasting, filtering still lacks a foundation-model paradigm shift. Classical methods from Gaussian filters (Li et al., 2015; Ito & Xiong, 2000; Cao et al., 2026), particle filters (Godsill, 2019) to PDE-based filters (Wang et al., 2025; Yau & Niu, 2025) instantiate a predict-then-update recursion, their strong performances usually depend on exactly known system-specific transition models, observation mappings, and noise assumptions. Learning-based filters reduce this modeling burden by training neural estimators from trajectories (Revach et al., 2022; Dahal et al., 2024), but they are still typically tied to one specific dynamical system (Ji et al., 2022), or to a parametric family sharing the same underlying dynamics (Holtmann et al., 2025). This raises a natural foundation-model question:

Can one model learn to filter across heterogeneous dynamical systems?

The challenges lie in structural heterogeneity rather than data scale alone: dynamical systems differ in their governing equations (Brunton et al., 2016), observation processes, physical units, and noise levels (Zheng et al., 2025), so a reusable filter cannot rely on a fixed state-observation interface. Existing foundation-inspired attempts only partially address this issue. LLMFilter (Liu et al., 2025) adapts a frozen LLM with system prompts, but remains trained on single datasets and flattens time and variable channels into tokens with unclear semantics. FM-UKF (Holtmann et al., 2025) inserts a time-series foundation model into a Bayesian filtering pipeline, but still requires a specified observation map and is evaluated mainly within one parametric family.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

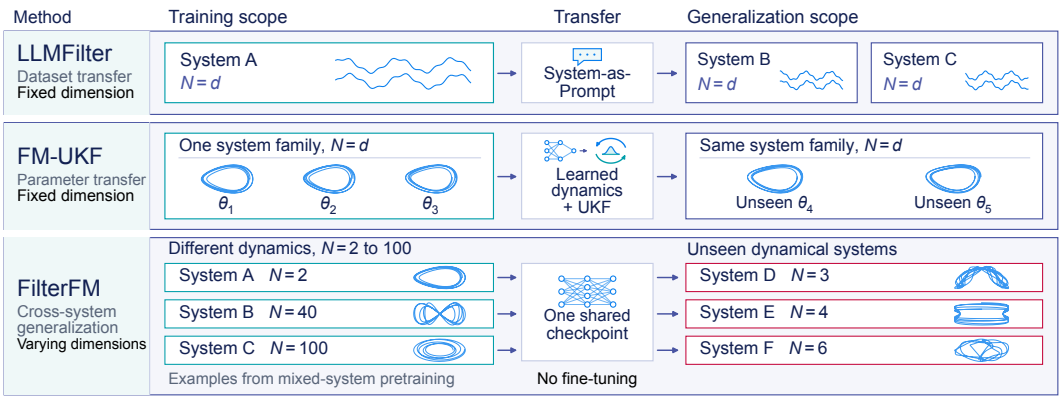

Figure 1: Generalization scopes of FilterFM, LLMFilter, and FM-UKF. LLMFilter trains on one system at a time and transfers across datasets at fixed state dimension N via System-as-Prompt. FM-UKF learns dynamics across parameter settings within one system family and uses a UKF with a known observation model. FilterFM is pretrained on a corpus of heterogeneous systems and evaluated on unseen systems using the same checkpoint without fine-tuning.

To fill this research gap, we introduce FilterFM, a transformer-based pretrained model for Bayesian filtering. Unlike LLMFilter and FM-UKF, FilterFM is pretrained across heterogeneous systems with varying dynamics and dimensions and is able to generalize to unseen systems with a single checkpoint. Figure 1 compares the three approaches' training and generalization scopes across datasets, system parameters, and state dimensions. Our model consists of a hierarchical two-level causal observation sequence encoder coupled with a prior-to-posterior state-reconstruction module. The encoder's design is motivated by two requirements: high-dimensional systems make dense attention over all time-variable pairs inefficient, and Bayesian filtering needs the current observation to update posterior rather than just coarse patch summaries. Accordingly, patch-level blocks compress completed patches into *causal history memory*, while intra-patch-level blocks preserve local evidence at each time step through *cross-scale conditioning*. This design retains the per-step resolution that Bayesian filtering requires for real-time correction based on the current observation. Within each scale, causal temporal attention first models each variable's temporal dependence. The resulting features are summarized into compact prototype memory and exchanged across variables through a gated reassembly router (Liu et al., 2026c). The state-reconstruction module then performs the filtering update: it first reads *the shifted local memory to form a prior*, then incorporates current observation to obtain a posterior, and decodes the posterior into the latent state estimate. Also to the best of our knowledge, FilterFM is the first model to support uncertainty estimation for Bayesian filtering, decoding multiple posterior quantiles to provide both a median state estimate and predictive uncertainty intervals.

To train FilterFM, we build a training dataset containing approximately 15K training trajectories, with about 45% from filtering numerical examples commonly used in previous literatures (Liu et al., 2025), 20% from synthetic low-dimensional chaotic systems (Lai et al., 2026), and 35% from high-dimensional composites formed by concatenating and orthogonally mixing low-dimensional trajectories. The resulting corpus spans system dimensions from 2 to 100, exposing the model to both standard filtering regimes and broad dimensional variation. On in-domain evaluations, FilterFM delivers strong gains along three practical axes: it achieves the lowest average raw-space RMSE for state-estimation accuracy, runs substantially faster than all compared baselines, and remains consistently more robust than LLMFilter and robust online Bayesian filters under observation-noise misspecification. Beyond in-domain performance, FilterFM demonstrates **zero-shot filtering on unseen heterogeneous systems**. FilterFM exhibits foundation-model-style scaling behavior: broader system coverage at an approximately matched training-data budget, more trajectories per system at fixed system diversity, and greater model capacity each improve generalization to unseen dynamics.

Our contributions are summarized as follows:

- We propose FilterFM, a pretrained model for Bayesian filtering built on a hierarchical two-level causal encoder and a prior-to-posterior state-reconstruction module. The model enables a single checkpoint to filter across heterogeneous dynamical systems with varying

dimensions and, to our knowledge, is the first Bayesian filtering model to introduce quantile prediction, providing both point estimates and predictive uncertainty intervals.

- We curate a heterogeneous training corpus spanning diverse dynamics and dimensions, and establish a comprehensive evaluation showing FilterFM's advantages in state-estimation accuracy, runtime efficiency, and robustness to observation-noise misspecification, together with the first demonstration of zero-shot filtering on unseen heterogeneous systems.
- We investigate whether scaling pretraining can improve generalization to unseen dynamical systems. This is the first systematic study of scaling behavior in Bayesian filtering across training-system diversity, training-trajectory coverage, and model capacity. Our results provide evidence that expanding pretraining can improve a single model's ability to estimate states across heterogeneous dynamics.

2 Related Work

Online Bayesian filtering. Classical Bayesian filters such as Gaussian filters or particle filters (Li et al., 2015; Ito & Xiong, 2000; Cao et al., 2026; Godsill, 2019) provide the *predict-then-update* paradigm for state estimation, which first predicts a prior distribution over the estimated state from the past observations and then updates it to a posterior via the transition model and observation map, but these methods require known transition models, observation maps, and noise statistics. In real-world applications, however, when explicit mathematical models are unavailable or noise statistics are misspecified, the feasibility of these approaches drops considerably.

Learning-based filtering. Neural filters reduce the need for explicit mathematical models by learning estimators from the trajectories. KalmanNet learns a neural Kalman gain correction (Revach et al., 2022) while other methods use different neural networks (Ji et al., 2022; Dahal et al., 2024) to map observation sequences to state estimates. LLMFilter (Liu et al., 2025) adopts a frozen LLM but suffers from patch-boundary inconsistencies across varying dimensions. Yet all these methods are system-specific and demand retraining for new dynamics, highlighting the need to broaden the generalization scope of Bayesian filtering toward reusable foundation-model-like frameworks.

Foundation models. Time-series foundation models (TSFMs) such as Chronos-2 (Ansari et al., 2025) and Moirai 2.0 (Liu et al., 2026b) demonstrate strong zero-shot forecasting capabilities, while Panda (Lai et al., 2026) and ChaosNexus (Liu et al., 2026a) target chaotic-system forecasting. In the area of state estimation, FM-UKF (Holtmann et al., 2025) inserts a causal transformer into an unscented Kalman filter but still requires an explicit observation map and operates within a single parametric family. Critically, forecasting and filtering follow *different information structures*: forecasting predicts a future window from an observed one, while online Bayesian filtering *causally updates* the state from *one new observation at a time*. FilterFM is designed for filtering: it learns the observation-to-state mapping from data, preserves strict per-step causality, and is the first pretrained model for Bayesian filtering across heterogeneous systems.

3 Problem Formulation

Let $\mathbf{x}_t \in \mathbb{R}^N$ denote the latent state of a dynamical system and $\mathbf{y}_t \in \mathbb{R}^M$ denote the noisy observation at time t. A Markovian state-space model (Weiss & Kalman, 1965) can be written as

$$\begin{aligned} \mathbf{x}_{t+1} &= f(\mathbf{x}_t) + \boldsymbol{\omega}_t, \\ \mathbf{y}_t &= h(\mathbf{x}_t) + \boldsymbol{\epsilon}_t, \end{aligned} \tag{1}$$

where f is the transition function, h is the observation function, and $\boldsymbol{\omega}_t \sim \mathcal{N}(0, \boldsymbol{R})$, $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{Q})$ denote process and observation noise sampled independently from Gaussian distributions. The filtering objective is to estimate the current state using only observations available up to the current time. Bayesian filtering expresses this objective through a prediction step and an update step:

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) &= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})\, d\mathbf{x}_{t-1}, \\ p(\mathbf{x}_t|\mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{\int p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{y}_{1:t-1})\, d\mathbf{x}_t}. \end{aligned} \tag{2}$$

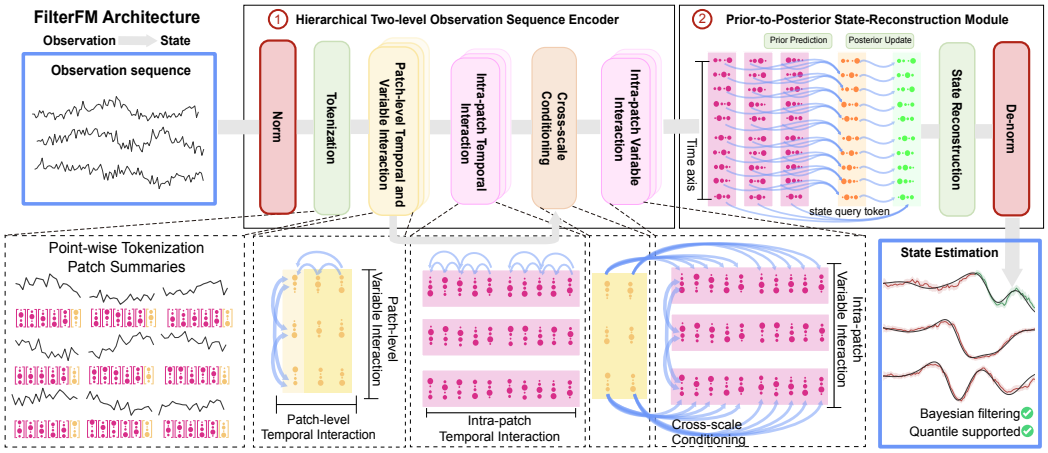

Figure 2: FilterFM architecture. The encoder combines patch-level temporal and variable interaction with intra-patch temporal interaction, cross-scale conditioning, and intra-patch variable interaction. State queries read shifted and current local memory for prior prediction and posterior update, followed by quantile decoding and de-normalization to the original state scale.

Here, $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1})$ is the prior distribution given past observations, and $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$ is the posterior distribution after incorporating $\mathbf{y}_t$. This prior-to-posterior update motivates our state-reconstruction module. A point estimate is the posterior mean $\hat{\mathbf{x}}_t = \mathbb{E}[\mathbf{x}_t|\mathbf{y}_{1:t}]$. Classical filters instantiate this recursion by specifying f, h, and the noise levels for each system. Learning-based filters instead replace the explicit Bayesian recursion with a neural estimator. We focus on the *foundation-model version* of this problem, using one neural model across different dynamical systems. For each system, the model receives $\mathbf{y}_{1:t} \in \mathbb{R}^{t\times M}$ and outputs $\hat{\mathbf{x}}_t \in \mathbb{R}^N$ according to the mapping $\hat{\mathbf{x}}_t = F_\theta(\mathbf{y}_{1:t})$.

4 Method

In this section, we introduce the architecture of FilterFM, which comprises a hierarchical two-level causal observation sequence encoder and a prior-to-posterior state-reconstruction module, as shown in Figure 2. We first describe how the encoder represents observation history and current evidence, then explain how these representations support prior prediction and posterior update.

4.1 Hierarchical Two-Level Observation Sequence Encoder

Our encoder is designed to preserve current observations for filtering while efficiently summarizing longer histories. We therefore represent observations at two temporal levels: patch-level embeddings summarize consecutive groups of observations, while intra-patch-level embeddings retain individual timesteps within each group. Causal temporal and prototype-based variable interaction operate at both levels, with cross-scale conditioning providing historical context to intra-patch embeddings.

Observation Tokenization and Patch Summaries. FilterFM first applies per-channel reversible instance normalization (Kim et al., 2022) to both observation and state sequences: $\tilde{y}_{t,m} = (y_{t,m} - \mu^y_m)/\sigma^y_m$ and $\tilde{x}_{t,n} = (x_{t,n} - \mu^x_n)/\sigma^x_n$, where μ^y_m, σ^y_m and μ^x_n, σ^x_n are the trajectory-wise mean and standard deviation of observation channel m and state channel n, respectively. We partition observations into patches of length L_p, with $t = pL_p + r$, $p \geq 0$, and $r = 1, \ldots, L_p$. A shared scalar encoder embeds normalized observations $\tilde{y}_{t,m}$, with channel and position embeddings added to form tokens $\mathbf{o}_{p,r,m} \in \mathbb{R}^d$. Causal temporal encoding and learned pooling with skip projections yield patch summaries $\mathbf{s}_{p,m} \in \mathbb{R}^d$, as detailed in Appendix D.2.

Temporal and Variable Interaction. To illustrate both levels, we describe one block and omit the layer indices distinguishing successive blocks for brevity. Let $\ell \in \{\text{patch}, \text{intra}\}$ denote the level, with $s = p$ or $s = (p, r)$, respectively. Its input and temporal output are $\mathbf{U}^\ell_s, \bar{\mathbf{U}}^\ell_s \in \mathbb{R}^{M\times d}$. The first

input stacks $\mathbf{s}_{p,m}^\top$ or $\mathbf{o}_{p,r,m}^\top$ as rows, while later inputs are the preceding block's full output $\mathbf{U}_s^{\ell,\text{out}}$ in Eq. (5).

Causal temporal interaction. Causally masked self-attention models temporal dependencies independently for each variable, followed by a residual feed-forward network (FFN):

$$\widetilde{\mathbf{U}}_s^\ell = \mathbf{U}_s^\ell + \text{SelfAttn}_\ell(\mathbf{U}_{\preceq_\ell s}^\ell), \bar{\mathbf{U}}_s^\ell = \widetilde{\mathbf{U}}_s^\ell + \text{FFN}_\ell(\widetilde{\mathbf{U}}_s^\ell), \tag{3}$$

where $\preceq_\ell$ includes patches up to p at the patch level and positions up to r within the same patch at the intra-patch level. Layer normalization, dropout, and projection biases are omitted for clarity.

Prototype-based variable interaction. To exchange information across different numbers of variables, we route the M tokens through a fixed number C of prototype slots (Liu et al., 2026c). Their input $\mathbf{V}_s^\ell \in \mathbb{R}^{M\times d}$ equals $\bar{\mathbf{U}}_s^\ell$ at the patch level and incorporates historical context at the intra-patch level through the cross-scale conditioning described below. The c-th rows $\mathbf{k}_c^{\ell,\pm} \in \mathbb{R}^d$ of the learnable matrices $\mathbf{K}_\pm^\ell \in \mathbb{R}^{C\times d}$ serve as the positive and negative attention keys for slot c, while projections $\mathbf{A}_q^\ell, \mathbf{A}_v^\ell \in \mathbb{R}^{d\times d}$ map each token $\mathbf{v}_{s,m}^\ell$ to query and value vectors:

$$\alpha_{s,m,c}^{\ell,\pm} = \text{softmax}_c\left(\frac{(\mathbf{A}_q^\ell \mathbf{v}_{s,m}^\ell)^\top \mathbf{k}_c^{\ell,\pm}}{\sqrt{d}}\right), \mathbf{p}_{s,c}^\ell = \frac{1}{M}\sum_{m'=1}^{M}\left(\alpha_{s,m',c}^{\ell,+} - \lambda_\ell \alpha_{s,m',c}^{\ell,-}\right)\mathbf{A}_v^\ell \mathbf{v}_{s,m'}^\ell. \tag{4}$$

The positive and negative attention branches normalize independently over the C slots for each variable, and λ_ℓ controls the contribution of negative attention. After obtaining the prototypes $\mathbf{p}_{s,c}^\ell$, we stack them into $\mathbf{P}_s^\ell \in \mathbb{R}^{C\times d}$ and apply self-attention followed by a residual FFN to obtain $\tilde{\mathbf{P}}_s^\ell$, then use this memory to update variable tokens through the gated reassembly router in Eq. (5):

$$\begin{aligned}
&\mathbf{R}_s^\ell = \text{CrossAttn}(\mathbf{V}_s^\ell, \tilde{\mathbf{P}}_s^\ell, \tilde{\mathbf{P}}_s^\ell),\ \mathbf{G}_s^\ell = \sigma(\text{MLP}_\ell([\mathbf{V}_s^\ell; \mathbf{R}_s^\ell])),\\
&\widehat{\mathbf{V}}_s^\ell = \mathbf{V}_s^\ell + \mathbf{G}_s^\ell \odot \left(\mathbf{R}_s^\ell (\mathbf{W}_o^{\ell,\text{proto}})^\top\right),\ \mathbf{U}_s^{\ell,\text{out}} = \widehat{\mathbf{V}}_s^\ell + \text{FFN}_\ell(\widehat{\mathbf{V}}_s^\ell).
\end{aligned} \tag{5}$$

Here $\mathbf{W}_o^{\ell,\text{proto}} \in \mathbb{R}^{d\times d}$ is a learnable output projection and $\widehat{\mathbf{V}}_s^\ell$ contains the updated variable representations, with $\mathbf{R}_s^\ell, \widehat{\mathbf{V}}_s^\ell, \mathbf{U}_s^{\ell,\text{out}} \in \mathbb{R}^{M\times d}$ and $\mathbf{G}_s^\ell \in (0,1)^{M\times d}$. The operators σ, $\odot$, and $[\cdot;\cdot]$ denote sigmoid, elementwise multiplication, and feature concatenation, respectively.

Cross-Scale Conditioning. To combine local evidence with longer-term context, intra-patch embeddings read completed patch summaries before the variable interaction described above. Stacked patch-level blocks yield summaries $\mathbf{H}_{p,m}^{\text{raw}}$, delayed by one patch as $\mathbf{H}_{0,m} = \mathbf{h}_m^{\text{bos}}$ and $\mathbf{H}_{p,m} = \mathbf{H}_{p-1,m}^{\text{raw}}$ for $p \geq 1$, with learned initial memory $\mathbf{h}_m^{\text{bos}}$. We define this history read through Retrieve, a gated cross-attention operator that updates a query $\mathbf{q} \in \mathbb{R}^d$ from a memory matrix $\mathbf{B} \in \mathbb{R}^{M\times d}$ containing one representation for each observation channel:

$$\mathbf{r} = \text{CrossAttn}(\mathbf{q}, \mathbf{B}, \mathbf{B}),\ \tilde{\mathbf{q}} = \mathbf{q} + \mathbf{g} \odot \mathbf{W}_o^{\text{ret}}\mathbf{r},\ \text{Retrieve}(\mathbf{q}, \mathbf{B}) = \tilde{\mathbf{q}} + \text{FFN}(\tilde{\mathbf{q}}), \tag{6}$$

where $\mathbf{r}, \tilde{\mathbf{q}} \in \mathbb{R}^d$ are the retrieved memory vector and updated query, respectively, and $\mathbf{g} = \sigma(\text{MLP}([\mathbf{q};\mathbf{r])}) \in (0,1)^d$. The temporal output then queries delayed patch memory to produce history-conditioned embeddings for intra-patch variable interaction:

$$\mathbf{v}_{p,r,m}^{\text{intra}} = \text{Retrieve}\left(\bar{\mathbf{u}}_{p,r,m}^{\text{intra}}, \mathbf{H}_{p,1:M}\right). \tag{7}$$

Within each intra-patch block, the history-conditioned embeddings $\mathbf{V}_{p,r}^{\text{intra}}$ undergo prototype-based variable interaction through Eqs. (4)–(5). The final block produces local memory $\mathbf{L}_{t,m}$, whose one-step shift $\mathbf{L}_{t,m}^- = \mathbf{L}_{t-1,m}$ provides past evidence for prior prediction with $\mathbf{L}_{0,m} = \mathbf{0}$ at initialization.

4.2 Prior-to-Posterior State-Reconstruction Module

The state-reconstruction module estimates latent states from the encoder's local memory through a prior-to-posterior update. For each state channel $n = 1, \ldots, N$, a learned query $\mathbf{q}_{t,n} \in \mathbb{R}^d$ combines channel identity, patch index, and intra-patch position to read the M observation tokens, allowing N to differ from M. Prior prediction and posterior update use the Retrieve structure in Eq. (6) with independently learned parameters to access shifted and current local memory, respectively.

Prior Prediction. The state query forms a prior token $\mathbf{z}_{t,n}^- \in \mathbb{R}^d$ from shifted local memory, using the encoded history before the current observation:

$$\mathbf{z}_{t,n}^- = \text{Retrieve}\left(\mathbf{q}_{t,n},\ \mathbf{L}_{t,1:M}^-\right). \tag{8}$$

Method	Ours	Learning-Based Filters							Online Bayesian Filters				
	FilterFM	LLMFilter	FM-UKF	MEstimator	RStateNet	ProTran	KalmanNet	RNNYYF	EnKF	PF	EnKFI	EnKFS	HubEnKF
Tracking	0.9425	1.9944	1.2556	2.7357	17.8542	*1.5862*	17.7382	18.7570	1.7846	1.7820	1.7858	1.7874	1.7853
Pendulum	0.0884	*0.0891*	0.1318	0.2234	0.2243	0.1095	0.1457	2.1915	0.0822	0.3266	0.2776	0.0933	0.0897
Selkov	0.0156	0.0189	*0.0165*	0.0264	0.0336	0.0182	0.1059	0.2291	0.0165	0.0167	0.1384	0.0217	0.0219
Oscillator	0.0164	0.0166	0.0171	0.0249	0.0320	0.0181	2.9264	0.1307	*0.0170*	0.0172	0.1313	0.0206	0.0212
Hopf	0.0157	0.0176	*0.0167*	0.0280	0.0304	0.0176	0.0540	0.1991	0.0164	0.0169	0.2095	0.0212	0.0216
Lorenz96	1.1781	*2.0539*	1.3314	2.6634	3.3996	2.5728	–	5.1669	5.3113	6.2552	5.3311	5.3875	5.3453
VL20	0.5376	2.4686	0.6877	2.2144	3.3186	2.1721	–	5.1090	1.4819	6.3915	5.9505	1.5404	*1.1619*
LorenzUV40	0.1215	0.2543	0.2487	*0.2525*	0.9310	0.2615	–	0.6085	0.3213	2.7444	0.3215	0.3142	0.3128
Vanderpol64	0.0715	0.3413	0.1422	0.2665	0.2236	0.2968	–	0.7162	0.1352	1.8032	*0.1346*	0.1361	0.1334
Rossler96	0.0592	0.2459	0.1560	0.2715	*0.1604*	0.2422	–	0.5554	0.1817	0.5458	0.1813	0.1880	0.1827
Duffing64	0.0236	0.0635	0.0353	0.0908	0.1157	0.0714	–	0.2743	*0.0446*	0.8295	0.0489	0.0474	0.0454
Burgers64	0.0218	0.0728	0.0294	0.0690	0.1285	0.0765	–	0.3352	*0.0308*	83.4534	2.8609	0.0410	0.0366
Fhn64	0.0187	0.0717	0.0322	0.0737	0.1003	0.0765	–	0.3272	*0.0341*	0.6590	0.0448	0.0366	0.0349
Cubic-Sensoring100	0.1404	0.7277	*0.1811*	0.6646	0.6316	0.7433	–	0.9869	0.1497	0.8148	0.9696	0.3281	0.3437
Average	0.2322	0.6026	0.3058	0.6861	1.9417	*0.5902*	4.1940	2.5419	0.6862	7.5469	1.3133	0.7117	0.6812
1st Count	13	0	0	0	0	0	0	0	1	0	0	0	0

Table 1: In-domain raw-space RMSE on 14 dynamical systems (lower is better). Red, blue underlining, and teal italics indicate the top three results; 1st Count denotes row-wise wins. Missing entries (–) are excluded from the corresponding method's average.

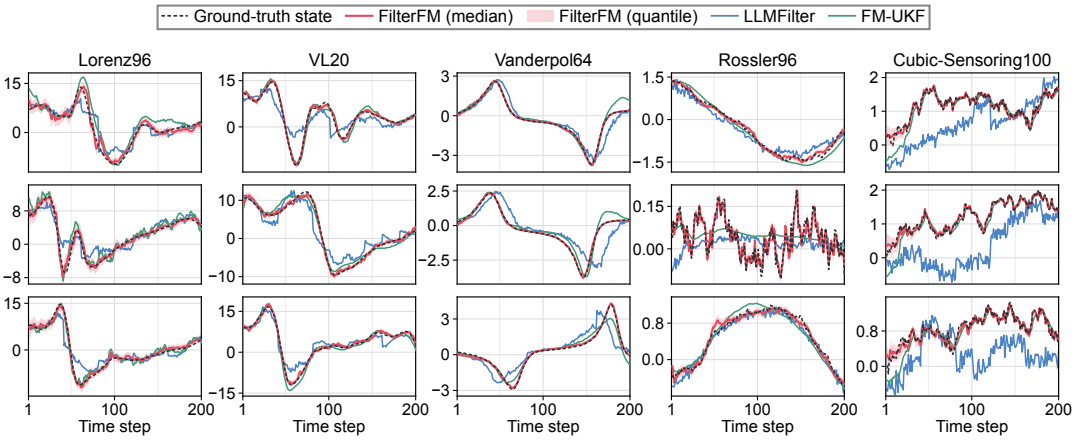

Figure 3: State-estimation trajectories on five high-dimensional in-domain systems (columns), with three state channels per system (rows). FilterFM tracks rapid changes in Lorenz96 and VL20, turning points in Vanderpol64, and small fluctuations in Rossler96. LLMFilter shows jumps and large offsets on Cubic-Sensoring100, while FM-UKF exhibits timing and amplitude errors. Light-red shading shows FilterFM's predicted 0.1–0.9 quantile interval.

Posterior Update. To incorporate the current observation, the prior token queries current local memory for evidence $\mathbf{e}_{t,n} \in \mathbb{R}^d$ to compute a gated innovation correction:

$$\mathbf{e}_{t,n} = \mathrm{Retrieve}(\mathbf{z}^-_{t,n}, \mathbf{L}_{t,1:M}),\ \mathbf{z}^+_{t,n} = \mathbf{z}^-_{t,n} + \mathbf{g}_{t,n} \odot \mathbf{\Delta}_{t,n}. \tag{9}$$

Two MLP branches take the concatenated prior, evidence, and their difference as input, producing the sigmoid gate $\mathbf{g}_{t,n} \in (0,1)^d$ and correction $\mathbf{\Delta}_{t,n} \in \mathbb{R}^d$, respectively.

State Reconstruction. A quantile head outputs $Q^+_{t,n,\tau} = W^+_\tau \mathbf{z}^+_{t,n}$ for $\tau \in \mathcal{Q} = \{0.1, 0.5, 0.9\}$, with the median mapped back to the state scale as $\hat{x}_{t,n} = \sigma^x_n Q^+_{t,n,0.5} + \mu^x_n$ and the outer quantiles describing uncertainty. Training supervises both prior and posterior with quantile losses, supplemented by a median Huber loss and prototype-key regularization, as detailed in Appendix D.6.

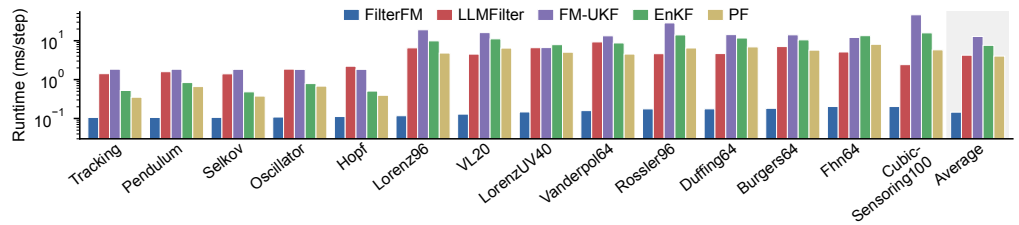

Figure 4: Per-step inference latency on 14 benchmark systems. The y-axis reports latency in ms/step on a base-10 logarithmic scale. Measurements use a single NVIDIA A800 GPU and exclude model loading. FilterFM is fastest on every system, averaging 0.145 ms/step.

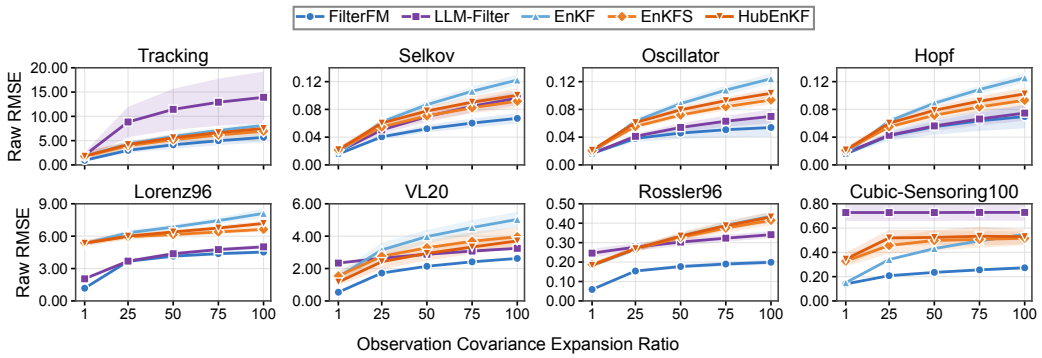

Figure 5: Observation-noise misspecification: true covariance is scaled up while online filters retain their original assumed covariance. FilterFM outperforms LLMFilter, EnKFS, and HubEnKF across low- and high-dimensional systems. Shading denotes standard deviation over 20 test trajectories.

5 Experiment

Benchmarks. We curate 14 established in-domain filtering tasks spanning low-dimensional nonlinear systems such as Selkov (Sel'kov, 1968) and Oscillator (Brunton et al., 2016), and high-dimensional chaotic systems such as Lorenz96 (Lorenz, 1995), VL20 (Vissio & Lucarini, 2020), and Cubic-Sensoring100 (Hazewinkel et al., 1983). We augment them with auxiliary systems from skew-product chaotic trajectories in Lai et al. (2026), using clean latent-state windows and per-channel Gaussian observation noise with a log-uniform scale. Because these source trajectories have at most 10 dimensions, we also form systems up to dimension 100 by normalizing, concatenating, and orthogonally mixing low-dimensional windows. In-domain evaluation uses 20 held-out trajectories per system, while OOS evaluation uses held-out systems with the same noise protocol. Full system definitions and generation details are in Appendix B.

Baselines and Metrics. We compare FilterFM with four baseline groups. Learning-based filters include MEstimator (Ji et al., 2022), RStateNet (Dahal et al., 2024), ProTran (Tang & Matteson, 2021), KalmanNet (Revach et al., 2022), and RNNYYF (Chen et al., 2025), each trained separately for each system. LLMFilter (Liu et al., 2025) uses the System-as-Prompt setting, whereas FM-UKF (Holtmann et al., 2025) couples a time-series foundation model with an unscented Kalman filter (Wan & Van Der Merwe, 2000). Online Bayesian baselines comprise EnKF, its robust variants EnKFI, EnKFS, and HubEnKF (Evensen, 1994), together with a particle filter (Godsill, 2019). We measure state-estimation accuracy by root mean square error (RMSE), defined in Appendix C, and runtime by wall-clock latency per filtering step (ms/step), excluding model loading.

5.1 In-Domain Performance in Terms of Accuracy, Runtime, and Robustness

FilterFM is pretrained on approximately 15K trajectories from the heterogeneous corpus described above, including 480 training trajectories from each in-domain system. The system-specific learning-based filters use the same 480 target-system trajectories for training. We adapt KalmanNet from its official implementation. For LLMFilter, we adapt the official implementation and train its input and output adapters on the same per-system data while keeping the LLaMA-7B (Touvron et al., 2023) backbone frozen, with the task prompts provided in Appendix F. For FM-UKF, we train a separate causal Transformer on pooled training trajectories from systems with the same state

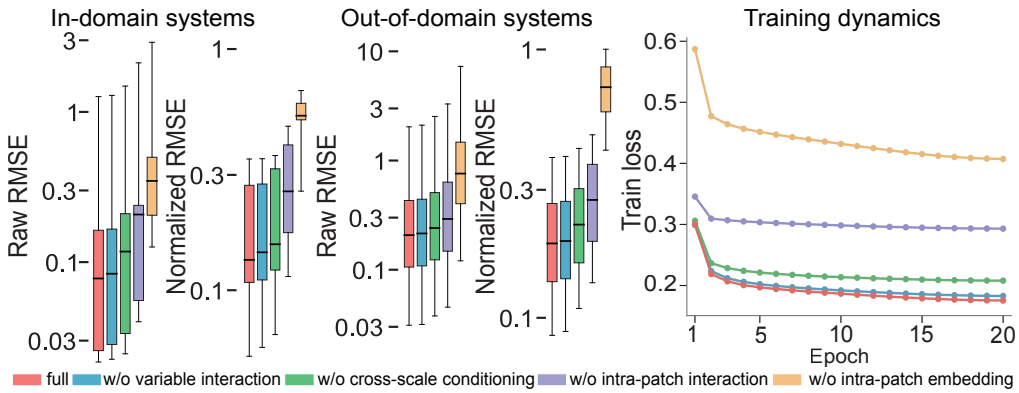

Figure 6: Ablation study of five architectural variants. Left and middle show raw and normalized RMSE distributions on in-domain and out-of-domain systems, respectively; boxes mark interquartile ranges and medians, with 5th–95th percentile whiskers. Right shows training-loss curves under identical data, optimization settings, and 20-epoch budgets. The full model achieves the lowest median errors on both splits and the lowest final training loss.

dimension. Online Bayesian filters use each system's ground-truth observation model. Appendix A provides implementation and training details.

Accuracy Analysis. Table 1 summarizes quantitative performance across all 14 systems. A single FilterFM checkpoint handles state dimensions from 2 to 100, wins 13 of 14 systems, and achieves the lowest average raw-space RMSE. Training on heterogeneous systems does not degrade per-benchmark accuracy, but rather *improves* it: FilterFM surpasses every per-system learning-based baseline, with the advantage most pronounced in high dimensions. Figure 3 visualizes representative trajectories comparing FilterFM with LLMFilter and FM-UKF on five high-dimensional systems, with three state channels shown per system. LLMFilter's autoregressive patch-wise decoding introduces **local discontinuities** and its flattening of time and variable channels loses cross-variable structure at scale. FM-UKF relies on an explicit observation model within a UKF; when model mismatch occurs, it incurs **systematic time lags and amplitude deviations**. In contrast, FilterFM closely tracks the ground truth while learning the observation-to-state mapping entirely from data without assuming an observation model. Appendix E.1 shows trajectories for all 14 systems.

Runtime. All runtime measurements are conducted on a single NVIDIA A800 GPU under the same hardware setting. Figure 4 compares per-step inference time. FilterFM runs at 0.145 ms/step on average, substantially faster than all compared baselines. LLMFilter's inference cost is **dominated by system-as-prompt tokens**, since patching contributes fewer than 20 tokens to the backbone. FM-UKF is slower because each step requires both **neural prediction and explicit UKF posterior correction**, incurring the computational cost of both stages. In contrast, FilterFM produces state estimates **end-to-end in a single forward pass**, eliminating prompt overhead and symbolic filtering.

Model Mismatch. We further evaluate the model's robustness under observation-noise misspecification (Liu et al., 2025; Evensen, 1994) by scaling the true observation covariance on test trajectories while keeping the assumed covariance fixed. Figure 5 shows that FilterFM consistently outperforms LLMFilter and robust online Bayesian filters (EnKF, EnKFS, HubEnKF) under increasing observation covariance expansion ratios (OCER $\in \{1, 25, 50, 75, 100\}$), on both low-dimensional (Selkov, Oscillator, Hopf) and high-dimensional (Lorenz96, VL20, Cubic-Sensoring100) systems. Appendix C.2 reports per-setting RMSE means and standard deviations over 20 test trajectories.

5.2 Zero-Shot Generalization and Ablation Analysis

FilterFM supports zero-shot filtering across heterogeneous dynamical systems. On held-out Panda systems (Lai et al., 2026) with three to six state variables, the same checkpoint transfers without fine-tuning, as illustrated in Figure 9 and the additional cases in Appendix E.2.

The ablations in Figure 6 identify current-observation access and intra-patch interaction as the main contributors to accuracy. **W/o intra-patch embedding** relies only on past patch representations and has the largest median RMSE increase on both splits, missing rapid changes in Figure 9. **W/o**

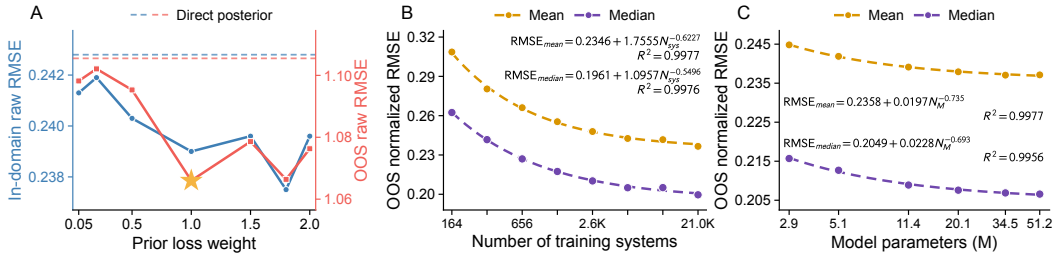

Figure 7: Prior-loss sweep and scaling behavior. (A) Raw RMSE on nine high-dimensional in-domain systems (blue; 40–100 state variables) and OOS systems (red). Dashed lines: direct posterior reconstruction; star: best OOS setting ($\lambda_{\text{prior}} = 1.0$). Full in-domain results can be found in Appendix C.3.2. (B,C) Mean and median OOS normalized RMSE versus training-system count and model capacity, with measurements (dots), power-law fits (dashed curves), and logarithmic x-axes.

intra-patch interaction retains current observations but omits their local processing, causing the second-largest increase. Removing **cross-scale conditioning** or **variable interaction** has smaller effects, with history conditioning contributing more. Under identical training settings and a 20-epoch budget, the full model also reaches the lowest training loss and the history-only variant the highest. Appendix C.3.1 provides further analysis of these ablations. In Figure 7A, explicit prior-to-posterior updates improve OOS performance over direct posterior reconstruction; we adopt a prior-loss weight of 1.0, which yields the lowest OOS RMSE.

5.3 Scaling Behavior

We study how training-system diversity, training-trajectory coverage, and model capacity affect Bayesian filtering on unseen dynamics. Improvements along all three axes suggest that a shared model can learn more general state-estimation capabilities from larger-scale pretraining.

Scaling system diversity. Increasing the nominal system count from 164 to 20,980 lowers mean OOS normalized RMSE from 0.3086 to 0.2366 and the median from 0.2624 to 0.1996 in Figure 7B. With approximately matched total training-trajectory counts, greater dynamical diversity improves OOS generalization, with diminishing gains as the number of training systems increases.

Scaling model parameters. We train six FilterFM models, ranging from 2.9M to 51.2M parameters, on the same expanded corpus of 301,873 long trajectories with a fixed number of training epochs. Table 10 details the changes in hidden widths, self-attention head counts, and encoder and state-reconstruction depths across model sizes. Mean OOS normalized RMSE decreases from 0.2448 to 0.2371 and the median from 0.2157 to 0.2066 in Figure 7C, indicating that greater capacity improves generalization with the training data held fixed.

Scaling training trajectories. With model capacity and training systems fixed, increasing the corpus from 20,471 to 301,873 trajectories lowers mean and median OOS normalized RMSE by 7.16% and 9.17%, respectively, averaged over three seeds. Both statistics decrease across all six dataset sizes, showing that broader coverage of initial conditions benefits generalization without changing the underlying training systems. Appendix C.3.5 details the experimental settings and results.

6 Conclusion and Future Work

We presented FilterFM, the first pretrained model for Bayesian filtering across heterogeneous dynamical systems with a single checkpoint. FilterFM combines a hierarchical two-level causal encoder with a prior-to-posterior state-reconstruction module. It delivers superior in-domain performance in state-estimation accuracy, runtime efficiency, and robustness to observation-noise misspecification, alongside zero-shot generalization to unseen dynamics. Ablation and scaling studies support the architectural design and the benefits of greater system diversity, trajectory coverage, and model capacity. Future work should expand training data to cover broader observation models, sensor modalities, and noise structures, advancing general-purpose filtering models.

AI USE STATEMENT

In this work, we used LLM-based assistants to polish the wording of text. We have not used generative AI tools to produce research ideas or experimental designs. No generative AI tool had access to non-public data. We have reviewed all AI-assisted work: all AI-assisted text was checked, corrected, and approved by the authors, and all AI-assisted code was verified for correctness, with every reported result produced by the authors' own runs. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work trains and evaluates FilterFM using synthetic trajectories and publicly available academic datasets used under their respective licenses; no private or personally collected data is involved, and no human-subject study was conducted. Our filtering datasets combine simulations of specified dynamical systems with transformations of publicly available Panda data (Lai et al., 2026), and we will document their provenance, construction procedures, and splits upon publication so that upstream restrictions remain traceable. The proposed method improves state estimation across heterogeneous dynamical systems and, to our knowledge, does not introduce risks beyond those already associated with Bayesian filtering, such as privacy concerns in tracking applications. As with any advance in state estimation, improved tracking capabilities could in principle be misused in surveillance settings; our training procedure neither targets nor specifically facilitates such uses.

REPRODUCIBILITY STATEMENT

We document everything needed to reproduce our results in the paper and its appendix. The construction pipeline of our filtering datasets, including benchmark dynamics, source datasets, observation generation, and train–test splits, is described in Appendices B.1–B.3; the protocols behind the runtime, robustness, ablation, and scaling experiments are specified in Appendices C.1–C.3; baselines, benchmarks, and the complete set of training and evaluation hyperparameters are listed in Appendices A, B.1, C, and D.7; the method itself is fully specified by Equations 3–9 and Appendices D.1–D.6. We will release the training code, the dataset construction scripts, the evaluation pipeline, the generated datasets, and the trained checkpoints upon publication.

REFERENCES

Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Guerron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. Chronos-2: From univariate to universal forecasting. *arXiv preprint arXiv:2510.15821*, 2025.

Timothy D. Barfoot. *State estimation for robotics*. Cambridge University Press, Cambridge, 2024.

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

J. M. Burgers. A mathematical model illustrating the theory of turbulence. In *Advances in applied mechanics*, volume 1, pp. 171–199. Academic Press, 1948.

Wenhan Cao, Tianyi Zhang, Zeju Sun, Chang Liu, Stephen S.-T. Yau, and Shengbo Eben Li. Nonlinear Bayesian filtering with natural gradient Gaussian approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 48(7):8581–8597, 2026.

Xiuqiong Chen, Zeju Sun, Yangtianze Tao, and Stephen S.-T. Yau. A uniform framework of Yau–Yau algorithm based on deep learning with the capability of overcoming the curse of dimensionality. *IEEE Transactions on Automatic Control*, 70(1):339–354, 2025.

Kevin Course and Prasanth B. Nair. State estimation of a physical system with unknown governing equations. *Nature*, 622(7982):261–267, 2023.

Pragyan Dahal, Simone Mentasti, Luca Paparusso, Stefano Arrigoni, and Francesco Braghin. RobustStateNet: Robust ego vehicle state estimation for autonomous driving. *Robotics and Autonomous Systems*, 172:104585, 2024.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proc. ICML*, Vienna, 2024.

Georg Duffing. *Erzwungene Schwingungen bei veränderlicher Eigenfrequenz und ihre technische Bedeutung*. Vieweg, Braunschweig, 1918.

Gerardo Duran-Martin, Matias Altamirano, Alex Shestopaloff, Leandro Sánchez-Betancourt, Jeremias Knoblauch, Matt Jones, Francois-Xavier Briol, and Kevin Patrick Murphy. Outlier-robust Kalman filtering through generalised Bayes. In *Proc. ICML*, Vienna, 2024.

Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5): 10143–10162, 1994.

Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6):445–466, 1961.

Simon Godsill. Particle filtering: The first 25 years and beyond. In *Proc. ICASSP*, Brighton, 2019.

Brian D. Hassard, Nicholas D. Kazarinoff, and Yieh-Hei Wan. *Theory and applications of Hopf bifurcation*. Cambridge University Press, Cambridge, 1981.

M. Hazewinkel, S. I. Marcus, and H. J. Sussmann. Nonexistence of finite-dimensional filters for conditional statistics of the cubic sensor problem. *Systems & Control Letters*, 3(6):331–340, 1983.

Sebastian H. M. Hickman, Ilija Trajković, Julia Kaltenborn, Francis Pelletier, Alex Archibald, Yaniv Gurwicz, Peer Nowack, David Rolnick, and Julien Boussard. Causal climate emulation with Bayesian filtering. In *Proc. NeurIPS*, San Diego and Mexico City, 2025.

Tobin Holtmann, David Stenger, Andres Posada-Moreno, Friedrich Solowjow, and Sebastian Trimpe. Sailing towards zero-shot state estimation using foundation models combined with a UKF. In *Proc. CDC*, Rio de Janeiro, 2025.

K. Ito and K. Xiong. Gaussian filters for nonlinear filtering problems. *IEEE Transactions on Automatic Control*, 45(5):910–927, 2000.

Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7(2):4630–4637, 2022.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *Proc. ICLR*, Conference held virtually, 2022.

Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

Jeffrey B. Lai, Anthony Bao, and William Gilpin. Panda: A pretrained forecast model for chaotic dynamics. In *Proc. ICLR*, Rio de Janeiro, 2026.

R. B. Levien and S. M. Tan. Double pendulum: An experiment in chaos. *American Journal of Physics*, 61(11):1038–1044, 1993.

Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. Kalman filter and its application. In *Proc. ICINIS*, Tianjin, 2015.

Chang Liu, Bohao Zhao, Jingtao Ding, and Yong Li. ChaosNexus: A foundation model for ODE-based chaotic system forecasting with hierarchical multi-scale awareness. In *Proc. ICML*, Seoul, 2026a.

Chenghao Liu, Taha Aksu, Juncheng Liu, Xu Liu, Hanshu Yan, Quang Pham, Silvio Savarese, Doyen Sahoo, Caiming Xiong, and Junnan Li. Moirai 2.0: When less is more for time series forecasting. *arXiv preprint arXiv:2511.11698*, 2026b.

Shiqi Liu, Wenhan Cao, Chang Liu, Zeyu He, Tianyi Zhang, Yinuo Wang, and Shengbo Eben Li. One filters all: A generalist filter for state estimation. In *Proc. NeurIPS*, San Diego and Mexico City, 2025.

Yiding Liu, Yifan Hu, Hongjie Xia, Peiyuan Liu, Hongzhou Chen, Xilin Dai, Zewei Dong, and Jiang-Ming Yang. Falcon-X: A time series foundation model for heterogeneous multivariate modeling. *arXiv preprint arXiv:2605.27286*, 2026c.

E. N. Lorenz. Predictability: A problem partly solved. In *Proc. ECMWF Seminar on Predictability*, Reading, 1995.

J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adrià López Escoriza, Ruud J. G. van Sloun, and Yonina C. Eldar. KalmanNet: Neural network aided Kalman filtering for partially known dynamics. *IEEE Transactions on Signal Processing*, 70:1532–1547, 2022.

Otto E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 1976.

E. E. Sel'kov. Self-oscillations in glycolysis 1. A simple kinetic model. *European Journal of Biochemistry*, 4(1):79–86, 1968.

Gerald L. Smith, Stanley F. Schmidt, and Leonard A. McGee. Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle. Technical Report NASA TR R-135, National Aeronautics and Space Administration, 1962.

Binh Tang and David S. Matteson. Probabilistic Transformer for time series analysis. In *Proc. NeurIPS*, Conference held virtually, 2021.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

B. van der Pol. On relaxation-oscillations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):978–992, 1926.

Gabriele Vissio and Valerio Lucarini. Mechanics and thermodynamics of a new minimal model of the atmosphere. *The European Physical Journal Plus*, 135(10):807, 2020.

E. A. Wan and R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proc. ASSPCC*, Lake Louise, 2000.

Yu Wang, Shuyuan Xu, Xueda Wei, Xinrui Luo, Stephen Shing-Toung Yau, Shing-Tung Yau, and Rongling Wu. Yau-YauAL: A computer tool for solving nonlinear filtering problems. *arXiv preprint arXiv:2506.08976*, 2025.

Zhenyu Wang and Leo Chiang. Monitoring chemical processes using judicious fusion of multi-rate sensor data. *Sensors*, 19(10):2240, 2019.

Leonard Weiss and R. E. Kalman. Contributions to linear system theory. *International Journal of Engineering Science*, 3(2):141–171, 1965.

Daniel S. Wilks. Effects of stochastic parametrizations in the Lorenz '96 system. *Quarterly Journal of the Royal Meteorological Society*, 131(606):389–407, 2005.

Shing-Tung Yau and Yi-Shuai Niu. An improved Yau-Yau algorithm for high dimensional nonlinear filtering problems. *Pure and Applied Mathematics Quarterly*, 21(6):2369–2423, 2025.

Jinlei Zhang, Shuai Mao, Lixing Yang, Wei Ma, Shukai Li, and Ziyou Gao. Physics-informed deep learning for traffic state estimation based on the traffic flow model and computational graph method. *Information Fusion*, 101:101971, 2024.

Donglei Zheng, Le Zhou, Yi Liu, and Qiang Liu. Dynamic process monitoring using total multirate linear Gaussian state space model. *IEEE Transactions on Industrial Informatics*, 21(6):4306–4315, 2025.

APPENDIX TABLE OF CONTENTS

A Baseline Details **15**

A.1 Classical and Robust Bayesian Filters 15

A.2 Learning-Based Filters 17

A.3 Other Baselines 18

A.4 Training and Evaluation Settings 18

B Dataset Details **19**

B.1 Benchmark Systems 19

B.2 Benchmark Split and Evaluation Protocol 23

B.3 Foundation Training Corpus Construction 23

C More Experiment Results **24**

C.1 Runtime Evaluation Details 25

C.2 Model Mismatch Evaluation Details 25

C.3 Additional Ablation and Scaling Results 26

C.3.1 Architectural Ablations 26

C.3.2 Prior-to-Posterior Updates and Prior-Loss Weight 27

C.3.3 Scaling the Diversity of Training Systems 29

C.3.4 Scaling Model Capacity 29

C.3.5 Scaling Training-Trajectory Coverage 30

C.3.6 Patch-Length Sensitivity 31

D Additional Method Details **32**

D.1 Representing Individual Observations 32

D.2 Summarizing Completed Patches 32

D.3 Temporal and Variable Interaction 33

D.4 Cross-Scale Conditioning and Local Memory 34

D.5 State Queries and State Reconstruction 35

D.6 Exact Training Loss 35

D.7 Main Model Training Configuration 36

D.8 Interpreting Patch-Level Interaction 37

D.9 Interpreting Prototype-Based Variable Interaction 40

E More State Estimation Showcase **40**

E.1 In-Domain State Estimation 40

E.2 Zero-Shot State Estimation 55

F System-as-Prompt Inputs for LLMFilter **71**

A Baseline Details

Our comparisons cover classical Bayesian filters, robust ensemble variants, task-specific neural estimators, and filters built around pretrained or jointly trained sequence models. This appendix introduces each baseline and details our benchmark adaptations and experimental settings for the results in Table 1. All methods receive identical test observations, and predictions are converted back to the original state coordinates before evaluation.

A.1 Classical and Robust Bayesian Filters

These filters implement the prior-to-posterior recursion in Eq. (2) using the known system functions and noise levels in Appendix B.1. We follow Section 3: $\mathbf{x}_t \in \mathbb{R}^N$ is the latent state, $\mathbf{y}_t \in \mathbb{R}^M$ is the observation, f and h are the transition and observation functions, and $\boldsymbol{R}$ and $\boldsymbol{Q}$ are the process- and observation-noise covariances.

- **Extended Kalman Filter (EKF).** As background for the ensemble methods below, the EKF extends the Kalman filter to nonlinear systems by applying first-order Taylor approximations to f and h. It approximates the posterior as $p(\mathbf{x}_t \mid \mathbf{y}_{1:t}) \approx \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, where $\boldsymbol{\mu}_t \in \mathbb{R}^N$ and $\boldsymbol{\Sigma}_t \in \mathbb{R}^{N\times N}$ are the posterior mean and covariance. The transition Jacobian at the previous posterior mean is defined as

$$\mathbf{F}_t := \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\boldsymbol{\mu}_{t-1}} \in \mathbb{R}^{N\times N}.$$

  The prediction step computes the prior mean and covariance as

$$\tilde{\boldsymbol{\mu}}_t = f(\boldsymbol{\mu}_{t-1}),\ \tilde{\boldsymbol{\Sigma}}_t = \mathbf{F}_t\boldsymbol{\Sigma}_{t-1}\mathbf{F}_t^\top + \boldsymbol{R}. \tag{10}$$

  Let $\mathbf{H}_t \in \mathbb{R}^{M\times N}$ be the Jacobian of h at the predicted mean $\tilde{\boldsymbol{\mu}}_t$. The update incorporates the current observation through

$$\begin{aligned}\mathbf{K}_t &= \tilde{\boldsymbol{\Sigma}}_t\mathbf{H}_t^\top\left(\mathbf{H}_t\tilde{\boldsymbol{\Sigma}}_t\mathbf{H}_t^\top + \boldsymbol{Q}\right)^{-1},\\ \boldsymbol{\mu}_t &= \tilde{\boldsymbol{\mu}}_t + \mathbf{K}_t\big(\mathbf{y}_t - h(\tilde{\boldsymbol{\mu}}_t)\big),\\ \boldsymbol{\Sigma}_t &= (\mathbf{I}_N - \mathbf{K}_t\mathbf{H}_t)\tilde{\boldsymbol{\Sigma}}_t.\end{aligned} \tag{11}$$

  The final state estimate is the mean of the resulting Gaussian posterior approximation, $\hat{\mathbf{x}}_t = \boldsymbol{\mu}_t$.

- **Ensemble Kalman Filter (EnKF).** The EnKF (Evensen, 1994) is an ensemble alternative to the EKF for nonlinear and chaotic systems. It approximates the posterior $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$ with N_e particles $\mathbf{x}_t^{(i)} \in \mathbb{R}^N$, where $i = 1, \ldots, N_\mathrm{e}$, and estimates covariances from the ensemble. The prediction step propagates the previous posterior particles through the transition model in Eq. (1) to obtain $\tilde{\mathbf{x}}_t^{(i)}$, whose ensemble approximates the prior $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1})$. In the update step, a perturbed predicted observation is sampled for each particle as

$$\mathbf{y}_t^{(i)} \sim \mathcal{N}\Big(h(\tilde{\mathbf{x}}_t^{(i)}), \boldsymbol{Q}\Big),\ i = 1, \ldots, N_\mathrm{e}. \tag{12}$$

  The Kalman gain and updated particles are then computed as

$$\begin{aligned}\mathbf{K}_t &= \mathrm{Cov}(\tilde{\mathbf{x}}_t^{(i)}, \mathbf{y}_t^{(i)})\big(\mathrm{Var}(\mathbf{y}_t^{(i)}) + 10^{-4}\mathbf{I}_M\big)^\dagger,\\ \mathbf{x}_t^{(i)} &= \tilde{\mathbf{x}}_t^{(i)} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{y}_t^{(i)}),\ i = 1, \ldots, N_\mathrm{e},\end{aligned} \tag{13}$$

  where Cov and Var denote the sample cross-covariance and sample covariance across ensemble members, respectively. Our implementation uses the diagonal regularizer and pseudoinverse $\dagger$ for numerical stability. Finally, averaging the updated ensemble gives the posterior state estimate:

$$\hat{\mathbf{x}}_t = \frac{1}{N_\mathrm{e}}\sum_{i=1}^{N_\mathrm{e}}\mathbf{x}_t^{(i)}. \tag{14}$$

- **Particle Filter (PF).** The bootstrap PF (Godsill, 2019) approximates the posterior $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$ with N_p weighted particles. It implements the prediction and update in Eq. (2) through sampling, likelihood weighting, and resampling.

  Sampling. Each previous posterior particle $\mathbf{x}_{t-1}^{(i)}$ is propagated according to the transition model, producing predictive particles $\tilde{\mathbf{x}}_t^{(i)}$ that approximate $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1})$:

$$\tilde{\mathbf{x}}_t^{(i)} \sim p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{(i)}) = \mathcal{N}\Big(f(\mathbf{x}_{t-1}^{(i)}), \boldsymbol{R}\Big)\,, \; i = 1, \ldots, N_\mathrm{p}. \tag{15}$$

  Weighting. Each predictive particle is assigned a likelihood measuring how well its predicted observation $h(\tilde{\mathbf{x}}_t^{(i)})$ agrees with the current observation $\mathbf{y}_t$. For Gaussian observation noise, these likelihoods are normalized to obtain the particle weights:

$$\begin{aligned} \tilde{w}_t^{(i)} &= \mathcal{N}\Big(\mathbf{y}_t; h(\tilde{\mathbf{x}}_t^{(i)}), \boldsymbol{Q}\Big)\,, \\ w_t^{(i)} &= \frac{\tilde{w}_t^{(i)}}{\sum_{k=1}^{N_\mathrm{p}} \tilde{w}_t^{(k)}}. \end{aligned} \tag{16}$$

  For numerical stability, we subtract the maximum log likelihood before exponentiation and normalization, preserving the normalized weights while preventing all unnormalized weights from underflowing to zero. The weighted particles approximate $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$ and give the state estimate:

$$\hat{\mathbf{x}}_t = \sum_{i=1}^{N_\mathrm{p}} w_t^{(i)} \tilde{\mathbf{x}}_t^{(i)}. \tag{17}$$

  Resampling. After computing $\hat{\mathbf{x}}_t$, we use systematic resampling to prepare the particle ensemble for the next prediction, allocating more particles to states with larger posterior weights. The procedure selects N_p copies from the weighted particles, allowing a particle to appear multiple times or be omitted. A particle with weight $w_t^{(i)}$ contributes $N_\mathrm{p} w_t^{(i)}$ copies on average, so its relative importance is carried forward through its number of copies. Each selected copy receives equal weight $1/N_\mathrm{p}$, and the resulting ensemble $\{\mathbf{x}_t^{(i)}\}_{i=1}^{N_\mathrm{p}}$ supplies the next prediction.

Robust EnKF variants. To reduce the influence of unusually large observation residuals, we implement innovation rejection, soft weighting, and Huber clipping, following the robust-update families discussed by Duran-Martin et al. (2024). All three variants retain the EnKF prediction, perturbed observations, and gain computation above. For the i-th particle and observation channel m, define the innovation and its standardized value as

$$d_{t,m}^{(i)} = y_{t,m} - y_{t,m}^{(i)}, \; u_{t,m}^{(i)} = \frac{d_{t,m}^{(i)}}{\sqrt{Q_{mm}}}.$$

Each variant replaces $\mathbf{d}_t^{(i)}$ with a modified innovation $\tilde{\mathbf{d}}_t^{(i)}$ in the common update

$$\mathbf{x}_t^{(i)} = \tilde{\mathbf{x}}_t^{(i)} + \mathbf{K}_t \tilde{\mathbf{d}}_t^{(i)}. \tag{18}$$

- **EnKFI.** This variant uses a hard decision shared by all ensemble members within each observation channel. It rejects the channel's innovation when the ensemble-averaged squared standardized residual exceeds c^2:

$$\tilde{d}_{t,m}^{(i)} = d_{t,m}^{(i)} \mathbb{I}\left[\frac{1}{N_\mathrm{e}} \sum_{k=1}^{N_\mathrm{e}} (u_{t,m}^{(k)})^2 \le c^2\right], \tag{19}$$

  where $\mathbb{I}[\cdot]$ is the indicator function. After the particle update in Eq. (18), EnKFI returns the ensemble mean $\hat{\mathbf{x}}_t = \frac{1}{N_\mathrm{e}} \sum_{i=1}^{N_\mathrm{e}} \mathbf{x}_t^{(i)}$ as the final state estimate.

- **EnKFS.** Soft weighting instead reduces each residual continuously, with a separate weight for every ensemble member and observation channel:

$$\tilde{d}_{t,m}^{(i)} = \frac{d_{t,m}^{(i)}}{\sqrt{1 + (u_{t,m}^{(i)}/c)^2}}. \tag{20}$$

  Small residuals are nearly unchanged, while larger residuals receive less weight. The final state estimate is the mean $\hat{\mathbf{x}}_t = \frac{1}{N_\mathrm{e}} \sum_{i=1}^{N_\mathrm{e}} \mathbf{x}_t^{(i)}$ of the ensemble updated by Eq. (18).

- **HubEnKF.** The Huberized update clips each standardized innovation at $\pm c$ and then restores its observation scale:

$$\tilde{d}_{t,m}^{(i)} = \sqrt{Q_{mm}}\,\mathrm{clip}(u_{t,m}^{(i)}, -c, c). \tag{21}$$

  This preserves innovations within the threshold and caps larger components at $c\sqrt{Q_{mm}}$. The final state estimate is the mean $\hat{\mathbf{x}}_t = \frac{1}{N_\mathrm{e}} \sum_{i=1}^{N_\mathrm{e}} \mathbf{x}_t^{(i)}$ of the ensemble updated by Eq. (18).

Shared implementation settings. To balance estimation accuracy and computational efficiency, we set $N_\mathrm{e} = N_\mathrm{p} = 1{,}000$. We initialize all particles from $\mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ and use $c = 5$ for the three robust variants. Diagonal noise variances have a numerical floor of 10^{-6}, and state particles are clipped componentwise to $[-100, 100]$ after propagation and correction. In the mismatch experiments of Appendix C.2, we increase the actual observation covariance while keeping the assumed $\boldsymbol{Q}$ fixed.

A.2 Learning-Based Filters

Following the learning-based baseline selection in LLMFilter (Liu et al., 2025), we evaluate MEstimator, RStateNet, ProTran, and KalmanNet, and additionally include RNNYYF. These task-specific models are trained separately on each system, with adaptations to their inputs, output dimensions, and supervised objectives described below. Each model predicts states from observations without receiving target states in its forward pass.

- **MEstimator.** The estimator of Ji et al. (2022) was developed alongside a control policy for legged locomotion. We adopt its multilayer-perceptron estimator and replace robot-specific inputs with causal observation history. At time t, we concatenate the standardized observation vectors from $t, t-2, t-4, t-6$ into a $4M$-dimensional input. This input passes through two fully connected hidden layers with 256 and 128 units, respectively, each followed by an ELU activation and dropout 0.1. An N-unit linear layer produces $\hat{\mathbf{x}}_t$, and training uses only state-estimation MSE.

- **RStateNet.** RobustStateNet (Dahal et al., 2024) combines learned motion prediction with a recurrent measurement update for ego-vehicle state estimation. We adapt it to our benchmark by replacing the vehicle-specific sensor branches with a single LSTM branch that processes the full M-dimensional observation vector together with the previous N-dimensional state estimate. We resize the output heads to predict an N-dimensional state increment and an $N \times M$ correction matrix, matching each system's state and observation dimensions. The current estimate is obtained by adding the predicted increment and a correction based on the observation residual to the previous estimate, and we train the model on each system with state-estimation MSE.

- **ProTran.** The probabilistic Transformer of Tang & Matteson (2021) models temporal structure through a recursively constructed latent representation. Our supervised filtering adaptation maintains a hidden-state history and lets the current query attend both to earlier hidden states and to observations up to the current time. We use representation width 256, eight attention heads, a 16-dimensional latent representation, and dropout 0.1. A latent projection updates the recurrent representation before state decoding. Training uses state-estimation MSE rather than the original variational objective, so this baseline evaluates an adaptation of ProTran's temporal architecture rather than its full probabilistic generative model.

- **KalmanNet.** KalmanNet (Revach et al., 2022) retains known transition and observation functions while learning the Kalman gain with recurrent networks. We adapt the gain-network architecture from the official KalmanNet repository[1], including its three GRUs for process, state, and innovation covariance features, and provide system-specific wrappers for the benchmark dynamics and observation maps. Our runs use input and output expansion factors of 5 and 1, respectively, and batch size 1. LLMFilter (Liu et al., 2025) reports training divergence with the official implementation on high-dimensional systems. Table 1 reports KalmanNet results for the five low-dimensional benchmark systems.

- **RNNYYF.** We adapt the recurrent Yau–Yau filtering framework of Chen et al. (2025) to learn state estimates directly, without explicitly representing the conditional density. Starting from a learned initial vector, the model updates a 512-dimensional hidden representation at each time

[1] https://github.com/KalmanNet/KalmanNet_TSP

step through a feed-forward network with $\tanh$ activations, using the previous representation and the observation increment $\mathbf{y}_t - \mathbf{y}_{t-1}$ as inputs. A linear layer then maps the updated representation to the N-dimensional state estimate $\hat{\mathbf{x}}_t$. We train the model on complete 200-step windows with unnormalized observations and target states, minimizing state-estimation MSE in the original state coordinates.

A.3 Other Baselines

These baselines use sequence models to reduce the amount of manually specified dynamics, but their interfaces and original generalization settings differ. We adapt both to the same benchmark trajectories and report their in-domain filtering performance, with the original model design and our changes distinguished below.

- **LLMFilter.** LLMFilter (Liu et al., 2025) maps observations into a frozen language model's embedding space and decodes its representations into state estimates, using System-as-Prompt to describe the filtering task. We build on the official repository[2], retain the frozen LLaMA-7B backbone[3], and train the observation encoder and state decoder separately for each system, using hidden width 512 and $\tanh$ activations in both adapters. The public implementation flattens time and observation channels before splitting them into fixed-length scalar segments, so patch boundaries need not coincide with time-step boundaries. Our adaptation preserves the time–channel layout by dividing each 40-step window into two contiguous 20-step patches, each containing all M observation channels and decoded into the corresponding N state channels. The encoder input and decoder output dimensions are therefore $20M$ and $20N$, respectively. Following the official preprocessing, we standardize observations and target states using training-set statistics. The public evaluation script reports RMSE in this standardized space, so we invert the state standardization before evaluation to compare all methods in the original state coordinates. Appendix F provides the System-as-Prompt inputs for all fourteen benchmark systems.

- **FM-UKF.** FM-UKF (Holtmann et al., 2025) combines a learned causal dynamics model with an unscented Kalman filter for posterior correction. Its official implementation[4] was developed for container-ship models with a 10-dimensional state space and 70 dynamics parameters, evaluating transfer to held-out parameterizations with a shared state layout. To adapt the predictor to our benchmark systems, we use a state-only causal Transformer and train one model per state-dimension group, pooling the systems within each group for in-domain evaluation. The adapted architecture uses four layers, width 256, eight attention heads, feed-forward width 1,024, and dropout 0.05, with input and output dimensions matched to each group. We train on standardized 80-step state windows with a next-step MSE loss at every position except the last under causal attention. During filtering, recent posterior state estimates provide temporal context for the learned dynamics prediction, and the UKF incorporates the current observation through the known observation map to obtain the posterior estimate.

A.4 Training and Evaluation Settings

Table 2 reports the training settings for the baselines in Table 1. Each task-specific baseline uses the 480 training trajectories of its target system, while FM-UKF pools those trajectories within each dimension group. Training uses AdamW with weight decay 10^{-5}, retaining the checkpoint with the lowest validation loss for evaluation. The 20 test trajectories for each system also serve as the validation set and are excluded from parameter updates. LLMFilter uses the cosine learning-rate scheduler provided in its official implementation, and we also apply cosine decay to the task-specific neural estimators. Our FM-UKF dynamics model uses a constant learning rate. Training stops after 10 consecutive epochs without improvement in validation loss or at the configured epoch limit.

[2] https://github.com/Anonymous-User-2367/LLMFilter
[3] https://huggingface.co/huggyllama/llama-7b
[4] https://github.com/Data-Science-in-Mechanical-Engineering/fm-ukf

Method	Window length	Batch size	Learning rate	Maximum training epochs
MEstimator	40	16	10^{-4}	50
RStateNet	40	16	10^{-4}	50
ProTran	40	16	10^{-4}	50
KalmanNet	40	1	10^{-4}	50
RNNYYF	200	16	4×10^{-4}	50
LLMFilter	40	1	10^{-4}	10
FM-UKF	80	128 ($N = 2, 4$) 16 ($N = 40, 64, 72$) 4 ($N = 96, 100$)	10^{-4}	50

Table 2: Baseline training settings for Table 1. Window lengths are measured in time steps, N denotes state dimension, and maximum training epochs is the preset upper limit for each run.

Preprocessing. MEstimator, RStateNet, ProTran, and KalmanNet standardize input observations using training-set statistics and minimize MSE in the original state coordinates. RNNYYF uses unnormalized observations and target states. Following the official implementations, LLMFilter standardizes observations and states, while FM-UKF standardizes the state sequences used to train its dynamics predictor. We evaluate all methods using RMSE in the original state coordinates. RNNYYF is less competitive on several systems in Table 1, and its validation loss fluctuates substantially on VL20 and Burgers64. Following the authors' formulation (Chen et al., 2025), our recurrent estimator takes the observation increment $\mathbf{y}_t - \mathbf{y}_{t-1}$ as input. Without normalization, differencing does not bring observation channels with different amplitudes and noise levels to a common scale, which may contribute to unstable optimization. These results suggest that unnormalized observation increments may not provide a robust default for neural estimation across heterogeneous systems.

B Dataset Details

During reproduction of LLMFilter (Liu et al., 2025), we found nearly constant observation residuals over long intervals in the Selkov, Oscillator, and Hopf datasets: after accounting for their one-step observation alignment, approximately 99.5% of adjacent residual vectors agree within 10^{-5}. We consider the resulting overly smooth observations unsuitable for evaluating filtering under independently sampled per-step noise. Since its public repository does not provide a dataset generator, we regenerate trajectories following the system definitions and simulation settings below, drawing fresh process and observation noise at each time step and aligning observations with their corresponding states. The original seven-system benchmark also includes only two high-dimensional systems, Lorenz96 and VL20, providing limited coverage of high-dimensional dynamics. We therefore expand the benchmark to fourteen systems and use it to evaluate all methods.

B.1 Benchmark Systems

This section documents the fourteen benchmark systems used in the main in-domain experiments. We summarize each system's characteristics and simulation settings in Table 3 before describing its dynamics and observation model.

These systems cover linear tracking, nonlinear oscillations, chaotic flows, and spatial coupling, providing diverse settings for state estimation. We numerically integrate the differential equations below using the methods and time steps in Table 3 to obtain the discrete-time state transition function f in Eq. (1). The filtering model includes additive process noise $\boldsymbol{\omega}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{R})$ and observation noise $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q})$. We use $\mathbf{I}_d$ for the $d \times d$ identity matrix and write the coupled dynamics componentwise, with the index ranges and boundary conditions specified for each system.

Tracking (Smith et al., 1962). This system is the classical constant-velocity kinematic model used in radar tracking, vision-based motion estimation, and navigation. The filtering task is to estimate

System	System type	State / Obs. Dim.	Δt	Integrator	$\boldsymbol{R}$	$\boldsymbol{Q}$
Tracking (Smith et al., 1962)	Linear target tracking	4/4	0.1	Forward Euler	$0.1\,\mathbf{I}_4$	$10\,\mathbf{I}_4$
Double-Pendulum (Levien & Tan, 1993)	Chaotic rigid-body mechanics	4/4	0.01	RK4	$10^{-4}\mathbf{I}_4$	$0.1\,\mathbf{I}_4$
Selkov (Sel'kov, 1968)	Biochemical glycolytic oscillation	2/2	0.01	RK4	$10^{-4}\mathbf{I}_2$	$10^{-3}\mathbf{I}_2$
Oscillator (Brunton et al., 2016)	Canonical nonlinear oscillation	2/2	0.01	RK4	$10^{-4}\mathbf{I}_2$	$10^{-3}\mathbf{I}_2$
Hopf (Hassard et al., 1981)	Normal-form bifurcation oscillator	2/2	0.01	RK4	$10^{-4}\mathbf{I}_2$	$10^{-3}\mathbf{I}_2$
Lorenz96 (Lorenz, 1995)	Chaotic atmospheric dynamics	72/72	0.01	RK4	$10^{-4}\mathbf{I}_{72}$	$10\,\mathbf{I}_{72}$
Vissio-Lucarini 20 (VL20) (Vissio & Lucarini, 2020)	Coupled climate surrogate dynamics	72/72	0.01	RK4	$10^{-4}\mathbf{I}_{72}$	$\mathbf{I}_{72}$
LorenzUV40 (Wilks, 2005)	Multiscale slow-fast atmosphere	40/40	0.005	RK4	$2.5\times10^{-7}\mathbf{I}_{40}$	$0.1\,\mathbf{I}_{40}$
Vanderpol64 (van der Pol, 1926)	Coupled nonlinear oscillator lattice	64/64	0.03	RK4	$1.8\times10^{-5}\mathbf{I}_{64}$	$0.1\,\mathbf{I}_{64}$
Rossler96 (Rössler, 1976)	Coupled chaotic flow cells	96/96	0.02	RK4	$10^{-3}\mathbf{I}_{96}$	$\mathbf{I}_{96}$
Duffing64 (Duffing, 1918)	Coupled bistable oscillators	64/64	0.05	RK4	$5\times10^{-5}\mathbf{I}_{64}$	$0.08\,\mathbf{I}_{64}$
Burgers64 (Burgers, 1948)	Viscous transport field	64/64	0.05	RK4	$1.25\times10^{-5}\mathbf{I}_{64}$	$0.05\,\mathbf{I}_{64}$
Fhn64 (FitzHugh, 1961; Nagumo et al., 1962)	Excitable reaction-diffusion lattice	64/64	0.05	RK4	$5\times10^{-5}\mathbf{I}_{64}$	$0.08\,\mathbf{I}_{64}$
Cubic-Sensoring100 (Hazewinkel et al., 1983)	Nonlinear sensing stress test	100/100	0.01	Forward Euler	$10^{-2}\mathbf{I}_{100}$	$10^{-2}\mathbf{I}_{100}$

Table 3: Configuration of the fourteen in-domain benchmark systems. Here $\boldsymbol{R}$ and $\boldsymbol{Q}$ denote the process and observation noise covariances used for testing, with 20 test trajectories per system.

the target's position and velocity from noisy observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$, with dynamics

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}, \mathbf{x} = [p_x, p_y, v_x, v_y]^\top.$$

Pendulum (Levien & Tan, 1993). This four-dimensional benchmark models a planar double pendulum and tests whether a filter can track strongly coupled angular motion whose errors grow rapidly in the chaotic regime. We use the state $\mathbf{x} = [\theta_1, \omega_1, \theta_2, \omega_2]^\top$, unit masses and lengths, $\delta = \theta_2 - \theta_1$, and $D = 1 + \sin^2\delta$. The dynamics are

$$\begin{cases} \dot{\theta}_1 = \omega_1, \\ \dot{\omega}_1 = \dfrac{\omega_1^2 \sin\delta\cos\delta + g\sin\theta_2\cos\delta + \omega_2^2\sin\delta - 2g\sin\theta_1}{D}, \\ \dot{\theta}_2 = \omega_2, \\ \dot{\omega}_2 = \dfrac{-\omega_2^2\sin\delta\cos\delta + 2g\sin\theta_1\cos\delta - 2\omega_1^2\sin\delta - 2g\sin\theta_2}{D}. \end{cases}$$

Here $g = 9.8$, and the observation model is $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

Selkov (Sel'kov, 1968). The Selkov model describes glycolytic oscillations in biochemical reaction networks and is a standard low-dimensional model for metabolic feedback. It captures how non-

linear reaction terms generate self-sustained oscillatory behavior, making it a useful benchmark for filtering biochemical dynamics from noisy concentration measurements:

$$\dot{\mathbf{x}} = \begin{bmatrix} -x_1 + ax_2 + x_1^2 x_2 \\ b - ax_2 - x_1^2 x_2 \end{bmatrix}, \mathbf{x} = [x_1, x_2]^\top,$$

with $a = 0.08$, $b = 0.6$, and observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

Oscillator (Brunton et al., 2016). This benchmark is a compact nonlinear oscillator with cubic restoring terms. Although low-dimensional, it introduces nontrivial state coupling and strongly nonlinear drift, making it a useful stress test for whether a filter can recover smooth latent motion from corrupted observations without relying on a linear update rule:

$$\dot{\mathbf{x}} = \begin{bmatrix} -ax_1^3 + bx_2^3 \\ -bx_1^3 - ax_2^3 \end{bmatrix}, \mathbf{x} = [x_1, x_2]^\top,$$

with $a = 0.1$, $b = 2.0$, and observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

Hopf (Hassard et al., 1981). The Hopf normal form models the onset of self-sustained oscillation as a stable equilibrium loses stability. With applications in fluid instabilities, chemical reactions, and electrical circuits, it provides a benchmark for recovering oscillatory states from noisy observations:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mu x_1 + \omega x_2 - ax_1 \|\mathbf{x}\|_2^2 \\ -\omega x_1 + \mu x_2 - ax_2 \|\mathbf{x}\|_2^2 \end{bmatrix}, \mathbf{x} = [x_1, x_2]^\top,$$

with $\mu = 0.5$, $\omega = 1.0$, $a = 1.0$, and observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

Lorenz96 (Lorenz, 1995). The Lorenz96 system, introduced by Lorenz, is a standard surrogate for atmospheric transport and weather predictability on a periodic domain. It is widely used in data assimilation because it combines simple local interactions with strongly chaotic global evolution, creating a representative medium-scale benchmark for online state reconstruction. Observations follow $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$, and the dynamics are

$$\dot{x}_j = \big(x_{j+1} - x_{j-2}\big)x_{j-1} - x_j + F,\ j = 1, \ldots, 72.$$

Here x_j denotes the j-th atmospheric variable, $F = 8$ is the external forcing, and the system uses periodic boundary conditions $x_{73} = x_1$, $x_0 = x_{72}$, and $x_{-1} = x_{71}$.

Vissio-Lucarini 20 (VL20) (Vissio & Lucarini, 2020). The Vissio-Lucarini 20 model extends the Lorenz framework with an additional temperature-like field and provides a minimalist but expressive benchmark for coupled climate dynamics. Relative to Lorenz96, it introduces direct interaction between circulation-like and thermodynamic variables, so filtering involves recovering both within-field evolution and cross-field exchange:

$$\begin{aligned} \dot{\phi}_j &= \big(\phi_{j+1} - \phi_{j-2}\big)\phi_{j-1} - \gamma\phi_j - \epsilon\theta_j + F, \\ \dot{\theta}_j &= \phi_{j+1}\theta_{j+2} - \phi_{j-1}\theta_{j-2} + \epsilon\phi_j - \gamma\theta_j + G. \end{aligned}$$

Here $\mathbf{x} = [\boldsymbol{\phi}^\top, \boldsymbol{\theta}^\top]^\top$ stacks the circulation-like variables ϕ_j and temperature-like variables θ_j for $j = 1, \ldots, 36$, with $F = 10$, $G = 0$, $\gamma = 1$, and $\epsilon = 1$. The model uses the same periodic boundary convention as Lorenz96. The observation model is $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

LorenzUV40 (Wilks, 2005). This is a multiscale slow-fast chaotic model with eight slow atmospheric variables and 32 fast variables. The coupling between these groups makes it a benchmark for filtering across distinct temporal scales. We use $\mathbf{x} = [\mathbf{u}^\top, \mathbf{v}^\top]^\top$, where $\mathbf{u} \in \mathbb{R}^8$ and $\mathbf{v} \in \mathbb{R}^{32}$, with observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$. Each slow variable u_i is coupled to the four fast variables with indices $j = 4i - 3, \ldots, 4i$. For $i = 1, \ldots, 8$ and these associated fast indices j, the dynamics are

$$\begin{cases} \dot{u}_i = (u_{i-1} - u_{i+2})u_{i+1} - u_i + 20 - \dfrac{hc}{b}\displaystyle\sum_{k=1}^{4} v_{4(i-1)+k}, \\ \dot{v}_j = cb(v_{j-1} - v_{j+2})v_{j+1} - cv_j + \dfrac{hc}{b}u_i. \end{cases}$$

The slow and fast indices wrap around their respective groups: $u_{i+8} = u_i$ and $v_{j+32} = v_j$. We set the coupling parameter $h = 1$, the amplitude scale $b = 10$, and the time-scale parameter $c = 10$.

Vanderpol64 (van der Pol, 1926). This system is a lattice of coupled Van der Pol oscillators, a classical model for self-excited oscillation in circuits and distributed oscillatory media. Filtering this lattice requires recovering oscillatory motion governed by nonlinear damping and coupling between adjacent oscillators. We use $\mathbf{x} = [\mathbf{q}^\top, \mathbf{v}^\top]^\top$, where q_i and v_i are the position and velocity of oscillator i, respectively. Observations follow $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$. For $i = 1, \ldots, 32$,

$$\begin{cases} \dot{q}_i = v_i, \\ \dot{v}_i = \mu(1 - q_i^2)v_i - q_i + \kappa(q_{i+1} - 2q_i + q_{i-1}). \end{cases}$$

The term $q_{i+1} - 2q_i + q_{i-1}$ couples each oscillator to its two neighbors on a ring, with $q_0 = q_{32}$ and $q_{33} = q_1$. We set $\mu = 2.0$ and the coupling strength $\kappa = 0.15$.

Rossler96 (Rössler, 1976). This system is a chain of coupled Rössler cells and serves as a high-dimensional spatiotemporal chaos benchmark. It represents locally interacting chaotic subsystems whose collective behavior remains difficult to estimate online because the latent state mixes oscillatory motion, stretching, and local transport across many channels. Each cell i has three coordinates $r_{x,i}$, $r_{y,i}$, and $r_{z,i}$, and the 96-dimensional state stacks these triples in cell order, with observations $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$. For $i = 1, \ldots, 32$,

$$\begin{cases} \dot{r}_{x,i} = -r_{y,i} - r_{z,i} + \kappa(r_{x,i+1} - 2r_{x,i} + r_{x,i-1}), \\ \dot{r}_{y,i} = r_{x,i} + a r_{y,i}, \\ \dot{r}_{z,i} = b + r_{z,i}(r_{x,i} - c). \end{cases}$$

The cells interact through their r_x coordinates, with periodic boundaries $r_{x,0} = r_{x,32}$ and $r_{x,33} = r_{x,1}$. We set $a = 0.2$, $b = 0.2$, $c = 5.7$, and the coupling strength $\kappa = 0.08$.

Duffing64 (Duffing, 1918). This system is a coupled Duffing oscillator chain with local bistable dynamics. Duffing-type systems arise in nonlinear structural vibration and forced mechanical response, and their double-well geometry makes filtering sensitive to whether the estimator preserves the correct oscillatory regime over time. We use $\mathbf{x} = [\mathbf{q}^\top, \mathbf{v}^\top]^\top$, where q_i and v_i are the position and velocity of oscillator i, respectively. Observations follow $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$. For $i = 1, \ldots, 32$,

$$\begin{cases} \dot{q}_i = v_i, \\ \dot{v}_i = q_i - q_i^3 - \eta v_i + \kappa(q_{i+1} - 2q_i + q_{i-1}). \end{cases}$$

The neighbor-coupling term uses periodic boundaries $q_0 = q_{32}$ and $q_{33} = q_1$. We set the damping coefficient $\eta = 0.22$ and the coupling strength $\kappa = 0.18$.

Burgers64 (Burgers, 1948). This system is a discretized viscous Burgers field, included as a PDE-like transport benchmark. Burgers dynamics capture the competition between nonlinear advection and diffusion and are widely used as a simplified model for transport and shock-like structure formation. The filtering task is to recover the evolving field from noisy measurements at 64 grid points. We use $\mathbf{x} = \mathbf{u} \in \mathbb{R}^{64}$, where u_i is the field value at grid point i, with observations $\mathbf{y}_t = \mathbf{u}_t + \boldsymbol{\epsilon}_t$. We approximate the spatial derivatives by centered finite differences:

$$\dot{u}_i = -u_i \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \nu \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + 0.12 \sin\left(\frac{2\pi(i-1)}{64}\right).$$

Here $i = 1, \ldots, 64$, $\Delta x = 1$ is the spatial grid spacing, and $\nu = 0.08$ is the viscosity. The two difference quotients approximate the first and second spatial derivatives, respectively, and the sine term provides a fixed spatial forcing. Periodic boundaries give $u_0 = u_{64}$ and $u_{65} = u_1$.

Fhn64 (FitzHugh, 1961; Nagumo et al., 1962). This system is a FitzHugh–Nagumo reaction-diffusion lattice for excitable media. FitzHugh–Nagumo dynamics are a standard abstraction for neuronal and cardiac excitation, and the coupled lattice form produces traveling activity patterns that require the filter to reconstruct both local activation and spatial propagation. We use $\mathbf{x} = [\mathbf{u}^\top, \mathbf{v}^\top]^\top$, where u_i is the activation variable and v_i is the recovery variable at site i. For $i = 1, \ldots, 32$,

$$\begin{cases} \dot{u}_i = u_i - \dfrac{u_i^3}{3} - v_i + 0.7 + 0.08(u_{i+1} - 2u_i + u_{i-1}), \\ \dot{v}_i = 0.08(u_i + 0.7 - 0.8v_i) + 0.02(v_{i+1} - 2v_i + v_{i-1}). \end{cases}$$

The neighbor-difference terms describe diffusion of activation and recovery between adjacent sites, with coefficients 0.08 and 0.02, respectively. Both fields have periodic boundaries: $u_0 = u_{32}$, $u_{33} = u_1$, $v_0 = v_{32}$, and $v_{33} = v_1$. The observation model is $\mathbf{y}_t = \mathbf{x}_t + \boldsymbol{\epsilon}_t$.

Cubic-Sensoring100 (Hazewinkel et al., 1983). This benchmark stresses nonlinear observation handling rather than partial observability. The latent dynamics are comparatively mild, but the cubic sensor map distorts the observation space enough that accurate filtering depends on learning a robust inverse relation from noisy measurements to latent state. Let $J \in \mathbb{R}^{100\times100}$ be the upper-shift matrix with $J_{i,i+1} = 1$ for $i < 100$ and zeros elsewhere. Then

$$\dot{\mathbf{x}} = -0.5\mathbf{x} + \cos(\mathbf{x}) + 0.1J\mathbf{x},$$

where $\cos(\mathbf{x})$ acts elementwise and observations satisfy $y_{t,m} = x_{t,m}^3 + \epsilon_{t,m}$ for $m = 1, \ldots, 100$.

B.2 Benchmark Split and Evaluation Protocol

For each of the fourteen in-domain systems, we use 480 training trajectories and 20 test trajectories, each containing $T = 200$ time steps. On each system, FilterFM and the applicable baselines are evaluated on the same 20 test trajectories, with per-system raw RMSE reported in Table 1.

B.3 Foundation Training Corpus Construction

Beyond the fourteen benchmark systems, we include Panda-derived low-dimensional trajectories and high-dimensional composites to broaden dynamical diversity and dimension range (Lai et al., 2026). Table 4 gives the exact composition of the main model's 15,120 training trajectories and the corresponding held-out sets. The 620 held-out trajectories form the validation pool, including the 280 benchmark trajectories used for Table 1.

Data source	Training	Held-out	Total
In-domain benchmark (14 systems)	6,720	280	7,000
Panda low-dimensional (790 systems)	3,360	140	3,500
Panda high-dimensional composites	5,040	200	5,240
Total for training domains	**15,120**	**620**	**15,740**
Panda zero-shot (457 systems)	0	8,207	8,207

Table 4: Composition of the main training corpus and held-out evaluation sets. All trajectories contain 200 time steps.

In-Domain Benchmark Data. The fourteen systems contribute 6,720 of the 15,120 training trajectories in FilterFM's joint corpus. The resulting model is evaluated on all fourteen systems without system-specific fine-tuning. Per-system baselines, including LLMFilter, use the same 480 training trajectories from their target system, while FM-UKF pools systems with the same state dimension.

Panda Low-Dimensional Systems. We use trajectories from the Panda dataset[5]. To formulate filtering tasks from these clean trajectories, we use $T = 200$ consecutive states as $\mathbf{X} = \mathbf{x}_{1:T} \in \mathbb{R}^{T\times N}$ and observe each state coordinate with additive Gaussian noise whose scale is relative to the trajectory amplitude:

$$y_{t,m} = x_{t,m} + \epsilon_{t,m},\ \epsilon_{t,m} \sim \mathcal{N}(0, \alpha^2\sigma_m^2(\mathbf{X})),\ \alpha \sim \mathrm{LogUniform}(0.15, 0.75), \tag{22}$$

where $\sigma_m(\mathbf{X})$ is the standard deviation of state coordinate m along the trajectory. Given $\mathbf{X}$ and α, noise is sampled independently across time steps and coordinates, with observation covariance $\boldsymbol{Q}$ defined by $Q_{mm} = \alpha^2\sigma_m^2(\mathbf{X})$.

Panda High-Dimensional Composites. To extend training beyond low-dimensional dynamics, we synthesize high-dimensional filtering systems by combining trajectories from multiple Panda systems. For a target dimension $N \in \{24, 32, 40, 48, 56, 64, 72, 80, 88, 96\}$, we first normalize each

[5] https://huggingface.co/datasets/GilpinLab/skew40

source window channel-wise, then concatenate enough normalized blocks to exceed N, select N columns, and apply a fixed random orthogonal mixing matrix:

$$\tilde{\mathbf{X}}^{(b)}_{:,j} = \frac{\mathbf{X}^{(b)}_{:,j} - \mu^{(b)}_j}{\sigma^{(b)}_j + \delta}, \mathbf{Z} = \Pi_N \,\mathrm{concat}(\tilde{\mathbf{X}}^{(1)}, \ldots, \tilde{\mathbf{X}}^{(B)}), \mathbf{x}^{\mathrm{comp}}_{1:T} = \mathbf{Z}\mathbf{O}_N^\top, \mathbf{O}_N^\top \mathbf{O}_N = \mathbf{I}_N, \quad (23)$$

where $\delta > 0$ is a small numerical stabilizer, Π_N denotes column selection to dimension N, and $\mathbf{O}_N$ is an orthogonal mixing matrix. Observations are then generated with the same identity-plus-noise construction as Eq. (22), but with $\alpha \sim \mathrm{LogUniform}(0.20, 0.90)$. These composites provide the model with high-dimensional training systems whose variable count and cross-channel structure differ substantially from the benchmark systems.

Held-out Zero-shot Systems. For the zero-shot experiments in Section 5.2, we use 8,207 trajectories from 457 unseen Panda systems with three to six state variables. Each trajectory contains $T = 200$ consecutive states, and observations follow the same Gaussian noise model as Eq. (22). These systems are excluded from training and fine-tuning, allowing us to evaluate transfer to previously unseen dynamics under the same observation construction. Figure 8 summarizes the state dimensions of the benchmark and Panda systems together with the training-data allocation. The benchmark covers dimensions from 2 to 100, while the low-dimensional Panda systems are concentrated at dimension 3. The composites extend Panda-derived training data across ten dimensions from 24 to 96.

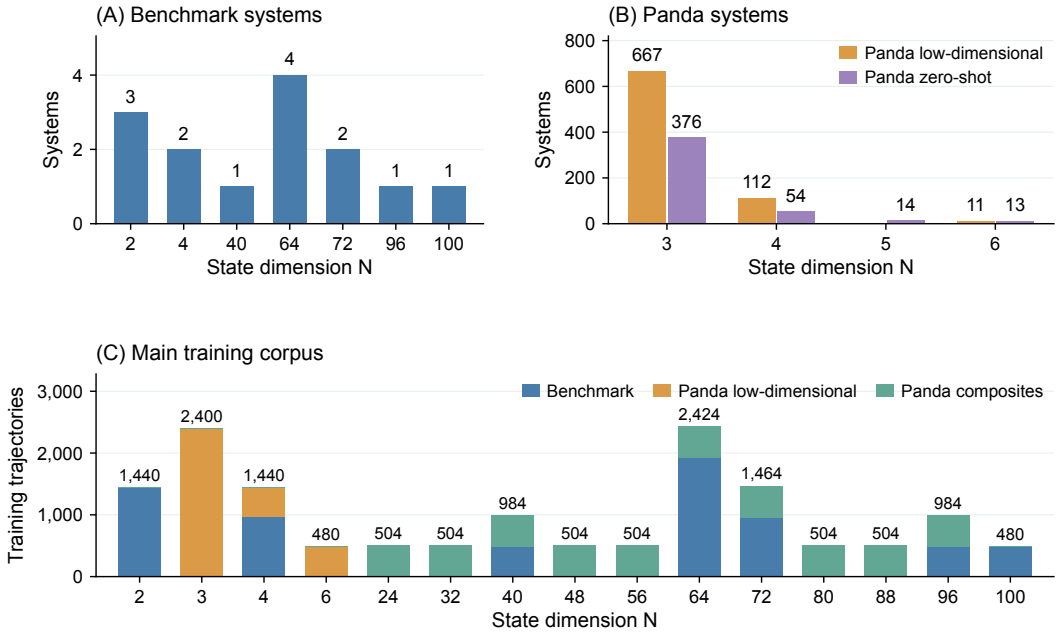

Figure 8: State-dimension distributions for (A) the fourteen benchmark systems and (B) the low-dimensional and zero-shot subsets in Table 4. (C) The 15,120 training trajectories, stacked by data source. All horizontal axes display state dimension N as discrete categories.

C More Experiment Results

This section extends the main experiments with per-system runtime comparisons, observation-noise misspecification results, and architectural ablations. We also report the prior-loss and patch-length sweeps and examine three aspects of scaling: the diversity of training systems, model capacity, and the number of training trajectories per system. Together, these experiments assess the computational cost, robustness, and transfer behavior of FilterFM and clarify how its components contribute to filtering accuracy.

Evaluation Metrics and Aggregation. We use root mean square error (RMSE) as the primary accuracy metric, following the squared-error criteria commonly used in Bayesian filtering and state estimation (Revach et al., 2022; Liu et al., 2025). Compared with mean absolute error (MAE), which is also common in time-series forecasting, RMSE gives greater weight to large estimation errors and is consistent with the minimum-mean-square-error criterion for state estimation. For an evaluation window of T time steps and N state variables, let $x_{t,n}$ and $\hat{x}_{t,n}$ denote the true state and its estimate in the original coordinates. We compute

$$\mathrm{RMSE}_{\mathrm{raw}} = \sqrt{\frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N}(\hat{x}_{t,n} - x_{t,n})^2},$$

$$\mathrm{RMSE}_{\mathrm{normalized}} = \sqrt{\frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N}\left(\frac{\hat{x}_{t,n} - x_{t,n}}{\sigma_n^x}\right)^2}.$$

Here σ_n^x is the standard deviation of the ground-truth state channel over the evaluated window, with a numerical floor of 10^{-6}, matching the instance normalization in Section 4.1. Only valid time steps and state channels contribute to the averages. Raw RMSE measures errors after reversing any state normalization, while normalized RMSE places channels with different amplitudes on a comparable scale. FilterFM's point estimate is its predicted posterior median.

In Table 1, each per-system score is the square root of the mean squared error over all 20 test trajectories, time steps, and valid state channels. The Average row is the arithmetic mean of the available per-system scores. For a long trajectory evaluated through multiple windows, we sum the squared errors and evaluated-coordinate counts across its windows before dividing and taking the square root. The resulting trajectory RMSEs are summarized by their mean or median. Window-level summaries instead apply these statistics directly to individual window RMSEs, as specified in the scaling experiments. In experiments with multiple seeds, we repeat this evaluation for each seed and report the mean and sample standard deviation of the resulting scores across seeds.

C.1 Runtime Evaluation Details

Table 5 reports the per-system runtime values used in Figure 4. All measurements are obtained on a single NVIDIA A800 GPU, exclude model loading time, and are averaged over the 20 held-out testing trajectories for each benchmark system. LLMFilter is timed with the full System-as-Prompt input protocol, including instruction and example tokens, to retain the input configuration used for its accuracy evaluation.

The runtime gap widens as the state dimension increases. EnKF and PF propagate 1,000 samples at every time step, so higher-dimensional systems increase the work required to evolve the samples and evaluate their observations. EnKF additionally forms covariance matrices and solves for the observation update, while PF evaluates particle likelihoods and resamples the population. These repeated operations make high-dimensional inference substantially more expensive: EnKF takes 0.4834–0.8383 ms per step on the five low-dimensional systems, compared with 7.9090–15.9068 ms on the nine high-dimensional systems. Sampling-based inference thus incurs a cost at every update even though no neural network is involved.

LLMFilter processes observation embeddings together with its System-as-Prompt tokens through the frozen LLaMA-7B backbone. Freezing the backbone reduces training requirements *but does not* remove this computation at inference. FM-UKF uses its neural dynamics model for prediction and still performs recursive UKF covariance propagation and posterior correction. This combination of neural prediction and recursive filtering makes FM-UKF the slowest method on average, and its cost grows with the number of sigma points in higher dimensions. FilterFM instead reconstructs states directly from learned observation representations without propagating an ensemble or solving a UKF update, giving the lowest measured runtime on all fourteen systems in Table 5.

C.2 Model Mismatch Evaluation Details

The model-mismatch experiment tests whether a trained estimator remains reliable when observations are noisier than expected. We keep the held-out state sequences unchanged and increase

Dataset	FilterFM	LLMFilter	FM-UKF	EnKF	PF
Tracking	0.1063	1.4186	1.8507	0.5247	0.3518
Pendulum	0.1062	1.5954	1.8457	0.8383	0.6614
Selkov	0.1067	1.4057	1.8379	0.4834	0.3767
Oscillator	0.1089	1.8586	1.8305	0.7892	0.6796
Hopf	0.1122	2.2081	1.8338	0.5068	0.3949
Lorenz96	0.1175	6.5480	19.0996	9.9052	4.8176
VL20	0.1294	4.5234	16.1513	11.1650	6.4611
LorenzUV40	0.1470	6.6081	6.6590	7.9090	5.0657
Vanderpol64	0.1592	9.3108	13.3544	8.7596	4.5340
Rossler96	0.1762	4.6360	28.6942	14.0968	6.4964
Duffing64	0.1766	4.6714	14.3440	11.7296	6.9267
Burgers64	0.1825	7.0708	14.1660	10.5132	5.7216
Fhn64	0.2030	5.1329	12.1294	13.4692	8.0933
Cubic-Sensoring100	0.2028	2.4218	46.4361	15.9068	5.8194
Average	**0.1453**	**4.2435**	**12.8738**	**7.6140**	**4.0286**

Table 5: Runtime details for the 14-system benchmark. All methods are evaluated on a single NVIDIA A800 GPU. Values are measured in ms/step with model loading excluded. LLMFilter timing includes the full System-as-Prompt input. Lower is better.

the Gaussian observation noise at each time step. For observation covariance expansion ratio $c \in \{1, 25, 50, 75, 100\}$, the test observations follow

$$\mathbf{y}_t^{(c)} = h(\mathbf{x}_t) + \boldsymbol{\epsilon}_t^{(c)}, \quad \boldsymbol{\epsilon}_t^{(c)} \sim \mathcal{N}(\mathbf{0}, c\mathbf{Q}).$$

Thus, c multiplies the noise covariance, while the corresponding noise standard deviation increases by $\sqrt{c}$. The underlying dynamics and target states remain the same, allowing the comparison to focus on observation corruption. We feed these observations to the trained models without further fitting, and the online Bayesian filters continue to use the nominal covariance $\mathbf{Q}$. Predictions are evaluated against the unchanged states in their original coordinates using raw-space RMSE. Table 6 reports the mean and standard deviation across the 20 test trajectories for each system and setting.

As the actual noise increases, a filter using the nominal covariance can place too much weight on unreliable observations. FilterFM retains lower mean errors than the compared methods across most of the reported settings, including the high-dimensional Lorenz96, VL20, and LorenzUV40 systems.

C.3 Additional Ablation and Scaling Results

We first examine which architectural components contribute to state estimation, then study the prior-loss weight, system diversity, model capacity, trajectory coverage, and patch length. The ablations share the main training corpus and optimization settings; the scaling experiments use the larger Panda-derived corpora described below.

C.3.1 Architectural Ablations

Figure 9 illustrates how the error differences in Figure 6 appear in individual trajectories. Removing intra-patch embeddings produces the largest median error increase on both in-domain and zero-shot systems. This variant estimates the current state from past patch representations without direct access to the current observation, which can leave substantial offsets and miss changes within a patch. Keeping the embeddings but removing intra-patch interaction gives the second-largest degradation: per-step observations remain available, but the model loses the local processing that combines them into a temporally coherent representation. The higher final training losses of these two variants indicate that their limitations are already visible when fitting the training tasks. These results suggest that Bayesian filtering accuracy depends strongly on incorporating current observations and that neglecting short-horizon temporal dependencies can substantially degrade performance.

Dataset	Method	OCER=1	OCER=25	OCER=50	OCER=75	OCER=100
Tracking	FilterFM	**0.943**±0.109	**3.011**±0.561	**4.143**±0.817	**4.977**±0.920	**5.672**±1.046
	LLM-Filter	1.994±0.672	8.839±3.140	11.438±4.237	12.902±4.844	13.921±5.307
	EnKF	1.785±1.020	4.300±0.582	5.865±0.564	7.094±0.608	8.134±0.677
	EnKFS	1.787±1.023	3.938±0.679	5.193±0.726	6.154±0.828	6.949±0.932
	HubEnKF	1.785±1.018	4.193±0.615	5.598±0.664	6.649±0.767	7.502±0.877
Selkov	FilterFM	**0.0156**±0.0010	**0.0405**±0.0036	**0.0521**±0.0054	**0.0603**±0.0066	**0.0671**±0.0074
	LLM-Filter	0.0189±0.0019	0.0505±0.0058	0.0696±0.0078	0.0845±0.0092	0.0963±0.0103
	EnKF	0.0165±0.0011	0.0619±0.0039	0.0868±0.0055	0.1061±0.0068	0.1223±0.0078
	EnKFS	0.0217±0.0043	0.0546±0.0046	0.0706±0.0064	0.0823±0.0080	0.0918±0.0095
	HubEnKF	0.0219±0.0041	0.0602±0.0048	0.0778±0.0066	0.0904±0.0083	0.1004±0.0097
Oscillator	FilterFM	**0.0164**±0.0009	**0.0382**±0.0049	**0.0459**±0.0084	**0.0507**±0.0112	**0.0539**±0.0130
	LLM-Filter	0.0166±0.0012	0.0412±0.0037	0.0540±0.0062	0.0630±0.0080	0.0699±0.0098
	EnKF	0.0170±0.0010	0.0631±0.0039	0.0883±0.0055	0.1078±0.0067	0.1242±0.0077
	EnKFS	0.0206±0.0025	0.0554±0.0033	0.0719±0.0044	0.0839±0.0053	0.0936±0.0061
	HubEnKF	0.0212±0.0023	0.0610±0.0032	0.0794±0.0044	0.0926±0.0054	0.1032±0.0063
Hopf	FilterFM	**0.0157**±0.0008	**0.0422**±0.0072	**0.0550**±0.0115	**0.0635**±0.0146	**0.0697**±0.0169
	LLM-Filter	0.0176±0.0014	0.0427±0.0046	0.0562±0.0071	0.0662±0.0092	0.0747±0.0111
	EnKF	0.0164±0.0009	0.0632±0.0033	0.0889±0.0048	0.1087±0.0059	0.1254±0.0069
	EnKFS	0.0212±0.0038	0.0549±0.0040	0.0715±0.0059	0.0835±0.0074	0.0932±0.0087
	HubEnKF	0.0216±0.0037	0.0602±0.0040	0.0787±0.0059	0.0919±0.0074	0.1023±0.0087
Lorenz96	FilterFM	**1.178**±0.026	**3.671**±0.139	**4.147**±0.158	**4.379**±0.170	**4.527**±0.176
	LLM-Filter	2.050±0.059	3.688±0.130	4.376±0.159	4.759±0.173	5.011±0.180
	EnKF	5.311±0.243	6.311±0.257	6.842±0.245	7.440±0.284	8.105±0.444
	EnKFS	5.388±0.241	5.944±0.268	6.150±0.197	6.389±0.212	6.631±0.219
	HubEnKF	5.345±0.298	6.003±0.258	6.394±0.202	6.753±0.219	7.175±0.281
VL20	FilterFM	**0.538**±0.011	**1.723**±0.034	**2.141**±0.044	**2.413**±0.051	**2.625**±0.055
	LLM-Filter	2.339±0.072	2.640±0.083	2.881±0.086	3.079±0.085	3.249±0.084
	EnKF	1.482±0.294	3.151±0.432	3.973±0.409	4.534±0.458	5.033±0.477
	EnKFS	1.540±0.292	2.756±0.388	3.282±0.454	3.686±0.573	3.961±0.586
	HubEnKF	1.162±0.194	2.422±0.475	2.936±0.435	3.328±0.440	3.706±0.373
LorenzUV40	FilterFM	**0.122**±0.006	**0.319**±0.009	**0.409**±0.013	**0.485**±0.019	**0.553**±0.026
	LLM-Filter	0.254±0.039	0.781±0.134	1.209±0.171	1.446±0.190	1.609±0.221
	EnKF	0.321±0.036	0.820±0.197	1.324±0.321	79.867±30.754	98.758±0.434
	EnKFS	0.314±0.038	0.510±0.071	0.603±0.075	1.975±1.189	3.871±0.777
	HubEnKF	0.313±0.037	0.514±0.067	1.590±0.950	3.825±1.190	4.196±0.041
Vanderpol64	FilterFM	**0.072**±0.003	**0.375**±0.015	0.564±0.018	0.653±0.019	0.713±0.023
	LLM-Filter	0.341±0.034	0.403±0.032	**0.454**±0.033	**0.497**±0.033	**0.534**±0.035
	EnKF	0.135±0.014	0.701±0.074	0.966±0.080	1.142±0.076	1.263±0.072
	EnKFS	0.136±0.013	0.522±0.040	0.698±0.054	0.818±0.066	0.918±0.074
	HubEnKF	0.133±0.015	0.518±0.039	0.681±0.052	0.816±0.072	0.901±0.072
Rossler96	FilterFM	**0.059**±0.001	**0.154**±0.006	**0.177**±0.010	**0.190**±0.012	**0.199**±0.012
	LLM-Filter	0.246±0.023	0.277±0.020	0.303±0.019	0.323±0.018	0.341±0.018
	EnKF	0.182±0.009	0.266±0.013	0.333±0.017	0.388±0.020	0.436±0.023
	EnKFS	0.188±0.009	0.268±0.015	0.327±0.019	0.375±0.023	0.415±0.026
	HubEnKF	0.183±0.009	0.269±0.015	0.334±0.020	0.387±0.024	0.432±0.027
Cubic-Sensoring100	FilterFM	**0.140**±0.005	**0.208**±0.006	**0.235**±0.008	**0.256**±0.010	**0.273**±0.013
	LLM-Filter	0.728±0.069	0.728±0.069	0.728±0.069	0.729±0.069	0.729±0.069
	EnKF	0.150±0.004	0.341±0.009	0.431±0.011	0.495±0.013	0.547±0.013
	EnKFS	0.328±0.044	0.457±0.069	0.498±0.083	0.507±0.068	0.516±0.058
	HubEnKF	0.344±0.053	0.519±0.067	0.523±0.061	0.533±0.078	0.529±0.060

Table 6: Model-mismatch raw-space RMSE on the 20 held-out testing trajectories, reported as mean ± trajectory-level standard deviation. FilterFM rows are shaded. Lower is better, and the best mean for each dataset and OCER is bolded.

Cross-scale conditioning uses historical patch summaries to help interpret noisy current observations, while variable interaction combines information across observation channels. Removing either module increases median RMSE on both in-domain and zero-shot systems, with a larger increase when cross-scale conditioning is removed. These increases are smaller than those caused by removing intra-patch embeddings or intra-patch interaction. This suggests that, for these filtering tasks, using recent observations and modeling short-horizon temporal dependencies contribute more to state-estimation accuracy than additional historical or cross-channel context.

C.3.2 Prior-to-Posterior Updates and Prior-Loss Weight

Table 7 compares direct posterior reconstruction with explicit prior-to-posterior updates. The direct variant reconstructs the posterior from current local memory without forming a prior. The In-domain raw column averages raw RMSE over the fourteen benchmark systems, with 20 test trajectories per

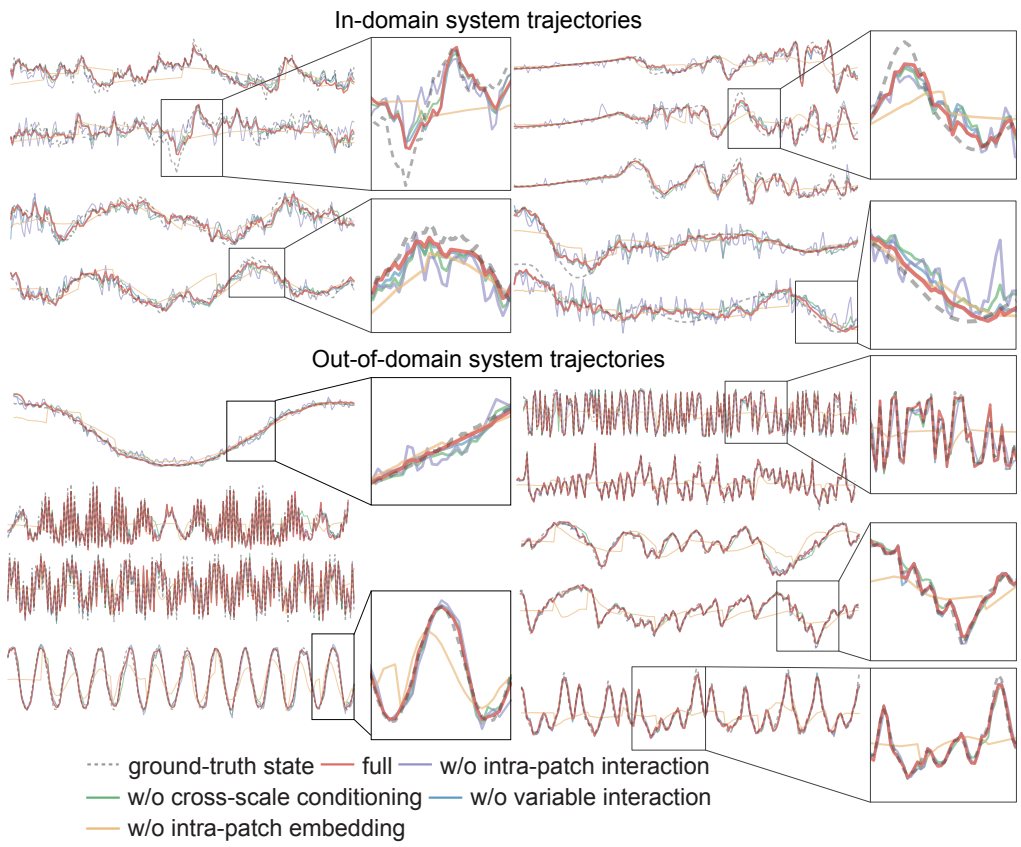

Figure 9: Ablation trajectories. In-domain and zero-shot system trajectories across five architectural variants.

system. High-dim raw reports the same metric for the nine systems with 40–100 state variables in Table 3, using the same 20 test trajectories per system for a total of 180. The OOS raw column evaluates transfer on 8,207 held-out Panda trajectories using a 200-step window per trajectory.

Every tested prior-loss weight improves all three scores relative to the direct update. A weight of 1.0 reduces OOS RMSE from 1.1055 to 1.0660, a relative improvement of 3.57%, and gives the best OOS result in the sweep. Increasing the weight to 1.8 gives the best in-domain scores but does not improve OOS accuracy. Our main model uses $\lambda_{\text{prior}} = 1.0$ because this setting achieves the lowest OOS raw RMSE in the sweep. This comparison suggests that supervising a prediction from past evidence provides a useful basis for incorporating the current observation.

Update rule	$\boldsymbol{\lambda_{\text{prior}}}$	**In-domain raw**	**High-dim raw**	**OOS raw**
Direct posterior	–	0.2351	0.2428	1.1055
Prior-to-posterior	0.05	0.2317	0.2413	1.0982
Prior-to-posterior	0.20	0.2334	0.2419	1.1021
Prior-to-posterior	0.50	0.2313	0.2403	1.0953
Prior-to-posterior	1.00	0.2317	0.2390	**1.0660**
Prior-to-posterior	1.50	0.2321	0.2396	1.0786
Prior-to-posterior	1.80	**0.2296**	**0.2375**	1.0664
Prior-to-posterior	2.00	0.2311	0.2396	1.0763

Table 7: Prior-loss sweep and direct-posterior ablation. In-domain raw averages over all fourteen benchmark systems; High-dim raw averages over the subset with state dimension ≥ 40; OOS raw averages over zero-shot systems. Lower is better.

C.3.3 Scaling the Diversity of Training Systems

We use the Panda dataset[6] to construct eight training subsets with progressively broader coverage of dynamical systems. We generate observations by adding independent Gaussian noise to each state trajectory at every time step, following the protocol in Appendix B.3. Across these subsets, the nominal system count ranges from 164 to 20,980, while the initial-condition count (IC), denoting the target number of trajectories per system, decreases from 128 to 1 to keep the total trajectory budget approximately matched. We independently initialize FilterFM for each subset and train it from scratch using the same architecture and 10-epoch schedule.

Table 8 lists the nominal system counts and the actual training-trajectory counts, which range from 19,101 to 20,702. The original dataset contains trajectories of 4,096 time steps, which we slice into 200-step windows for training and evaluation. Each trajectory yields 21 windows, including an end-aligned final window that overlaps the preceding window by 104 steps. Evaluation uses the same 172,599 windows from 8,219 held-out Panda trajectories, and Figure 7B summarizes normalized RMSE across these windows. The empirical fits are

$$\mathrm{RMSE}_{\mathrm{med}}(N_{\mathrm{sys}}) = 0.1961 + 1.0957 N_{\mathrm{sys}}^{-0.5496},\ R^2 = 0.9976,$$
$$\mathrm{RMSE}_{\mathrm{mean}}(N_{\mathrm{sys}}) = 0.2346 + 1.7555 N_{\mathrm{sys}}^{-0.6227},\ R^2 = 0.9977.$$

With approximately matched training-trajectory counts, broader dynamical coverage lowers both mean and median OOS RMSE. The gains diminish at larger system counts.

IC	N_{sys}	Train trajectories	OOS normalized median	OOS normalized mean
128	164	19,101	0.2624	0.3086
64	328	19,806	0.2417	0.2804
32	656	19,672	0.2270	0.2661
16	1312	19,534	0.2174	0.2554
8	2623	19,369	0.2102	0.2479
4	5245	19,357	0.2050	0.2426
2	10490	19,349	0.2051	0.2417
1	20980	20,702	**0.1996**	**0.2366**

Table 8: System-diversity scaling details. IC is the target number of initial conditions per system, and N_{sys} is the nominal system count of the corresponding training subset. The actual training-trajectory counts reflect data filtering and validation allocation. Values are OOS normalized RMSE statistics over 200-step windows sliced from 4096-step held-out trajectories. Lower is better.

C.3.4 Scaling Model Capacity

To examine how model capacity affects generalization with the training data held fixed, we train FilterFM models with six different parameter counts on an expanded Panda-derived corpus[7] containing 301,873 training trajectories from 20,471 systems, with 347 additional trajectories reserved for validation. The systems have three to six state variables and up to sixteen initial conditions each. We construct filtering tasks by adding independent Gaussian observation noise at each time step of the 4,096-step state trajectories and slicing the paired sequences into 200-step windows. All models share a 10-epoch training schedule, optimization settings, quantile levels $\mathcal{Q} = \{0.1, 0.5, 0.9\}$, and prior-loss weight $\lambda_{\mathrm{prior}} = 1.0$.

Table 10 specifies the model widths and depths. The 2.89M–34.54M models increase the hidden width d and FFN hidden width while keeping block counts fixed and using four heads for the encoder's temporal self-attention and self-attention among prototypes. The 51.21M model retains the widths of the 34.54M model, uses six heads for these self-attention operations, and adds one patch-summary temporal encoding layer, one patch-level block, one intra-patch block, and one prior-to-posterior state-reconstruction block.

We evaluate each model's epoch-10 checkpoint on the same 172,599 windows from 8,219 held-out trajectories. For each trajectory, raw RMSE is computed over all time steps and valid state coordinates across its windows, then summarized by the mean and median across trajectories. Table 9

[6] https://huggingface.co/datasets/GilpinLab/scalinglaw

[7] https://huggingface.co/datasets/GilpinLab/skew-mixedp-ic16

and Figure 7C show that increasing capacity from 2.89M to 51.21M parameters lowers training loss, in-domain validation RMSE, and OOS error. The OOS mean decreases from 0.8600 to 0.8104 and the median from 0.2189 to 0.2104, indicating better generalization to unseen systems.

Parameters	Training loss at epoch 10	In-domain validation raw RMSE	OOS raw RMSE	
			Mean	Median
2.89M	0.148945	0.284954	0.860030	0.218894
5.09M	0.143718	0.277112	0.847594	0.216160
11.36M	0.135006	0.266257	0.833914	0.213774
20.11M	0.129503	0.263754	0.829695	0.212346
34.54M	0.125071	0.257718	0.817275	0.210798
51.21M	**0.119767**	**0.254317**	**0.810427**	**0.210380**

Table 9: Model-parameter scaling training and raw-space evaluation results. All results use the same data, training budget, and epoch-10 checkpoint selection and are from single-seed runs. Lower is better; the best value in each metric column is bolded.

Parameters	Hidden width d	FFN hidden width	Self-attention heads	Patch-summary temporal encoding layers J_{enc}	Patch-level blocks	Intra-patch blocks	Prior-to-posterior state-reconstruction blocks
2,888,007	96	384	4	2	3	2	1
5,091,751	128	512	4	2	3	2	1
11,360,871	192	768	4	2	3	2	1
20,112,167	256	1024	4	2	3	2	1
34,541,847	336	1344	4	2	3	2	1
51,205,431	336	1344	6	3	4	3	2

Table 10: Architecture configurations for model-capacity scaling. Patch-summary temporal encoding layers process observation tokens before pooling, as described in Appendix D.2. Patch-level blocks combine causal temporal interaction with prototype-based variable interaction; intra-patch blocks additionally apply cross-scale conditioning between these operations, as detailed in Appendices D.3–D.4. Prior-to-posterior state-reconstruction blocks follow Section 4.2 and Appendix D.5. The listed head count applies to temporal self-attention for patch-summary construction, patch-level and intra-patch temporal self-attention, and self-attention among prototypes at both levels. Gated reassembly and the Retrieve operators use single-head cross-attention.

C.3.5 Scaling Training-Trajectory Coverage

To study whether denser trajectory coverage improves generalization at fixed system diversity, we keep the 20.11M-parameter FilterFM and the same 20,471 training systems while varying the maximum number of trajectories per system over $\{1, 2, 4, 8, 12, 16\}$. This produces six datasets containing 20,471 to 301,873 training trajectories. Each configuration is trained independently for 10 epochs across three seeds, with the same validation set, optimization settings, and evaluation protocol as the model-capacity scaling experiment. Table 11 reports the mean and sample standard deviation of each RMSE statistic across seeds.

Figure 10 and Table 11 show that increasing trajectory coverage improves OOS estimation when the set of training systems is fixed. Across the six dataset sizes, the seed-averaged normalized mean and median RMSE decrease monotonically from 0.25605 to 0.23773 and from 0.22841 to 0.20745, respectively, corresponding to relative reductions of 7.16% and 9.17%, with diminishing gains at larger sizes.

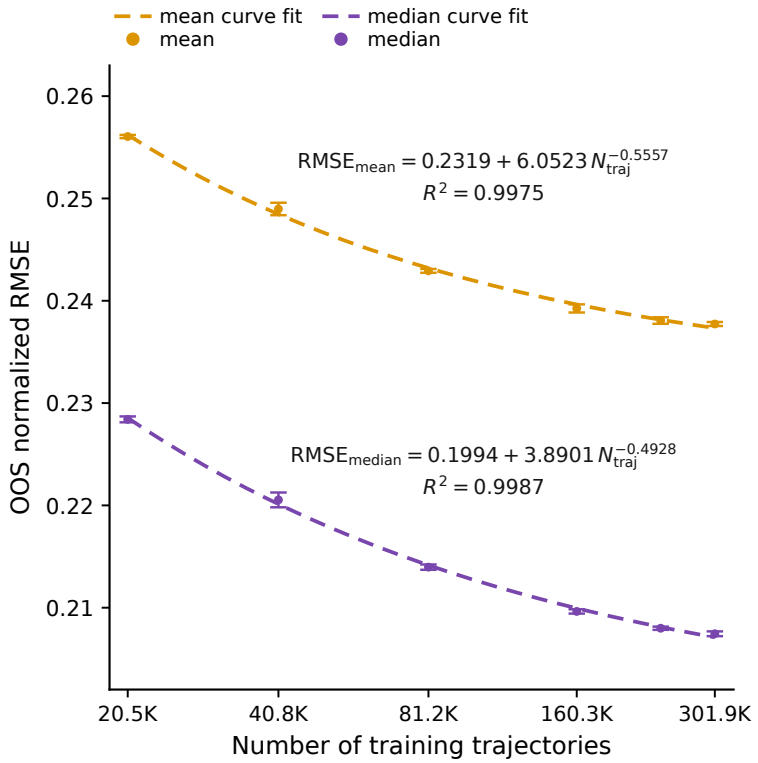

Figure 10: Training-trajectory scaling at fixed model capacity and system diversity. Points show OOS trajectory-level mean or median RMSE averaged over three seeds, with vertical error bars indicating one sample standard deviation. Dashed curves show power-law-with-floor fits.

Maximum trajectories per system	Training trajectories	OOS normalized RMSE		OOS raw RMSE	
		Mean	Median	Mean	Median
1	20,471	$0.25605_{\pm 0.00016}$	$0.22841_{\pm 0.00029}$	$0.90963_{\pm 0.00142}$	$0.23163_{\pm 0.00017}$
2	40,845	$0.24897_{\pm 0.00061}$	$0.22054_{\pm 0.00072}$	$0.87511_{\pm 0.00531}$	$0.22380_{\pm 0.00081}$
4	81,239	$0.24292_{\pm 0.00019}$	$0.21397_{\pm 0.00026}$	$0.84550_{\pm 0.00407}$	$0.21733_{\pm 0.00036}$
8	160,304	$0.23925_{\pm 0.00040}$	$0.20964_{\pm 0.00021}$	$0.83187_{\pm 0.00150}$	$0.21345_{\pm 0.00036}$
12	235,557	$0.23807_{\pm 0.00032}$	$0.20799_{\pm 0.00016}$	$\mathbf{0.82480}_{\pm \mathbf{0.00333}}$	$0.21252_{\pm 0.00026}$
16	301,873	$\mathbf{0.23773}_{\pm \mathbf{0.00020}}$	$\mathbf{0.20745}_{\pm \mathbf{0.00023}}$	$0.82621_{\pm 0.00731}$	$\mathbf{0.21234}_{\pm \mathbf{0.00032}}$

Table 11: Training-trajectory scaling across three seeds. Entries report the mean $\pm$ sample standard deviation of each OOS trajectory statistic. All runs use the same 20,471 training systems and 20.11M-parameter model. Lower is better; the best average in each metric column is bolded.

C.3.6 Patch-Length Sensitivity

We vary the temporal patch length P while keeping the 200-step window, training corpus, all other architectural hyperparameters, quantile objective, prior-loss weight ($\lambda_{\text{prior}} = 1.0$), and 20-epoch optimization budget fixed. Each result uses the checkpoint selected by validation raw RMSE.

Relative to the best result at $P > 1$ for each metric, $P = 1$ increases in-domain raw and normalized RMSE by 17.2% and 9.2%, and OOS raw and normalized RMSE by 12.0% and 6.0%, respectively. This trend is consistent with the gains from increasing input patch length reported for the time-series foundation model TimesFM (Das et al., 2024). At $P = 1$, patch-level attention still models dependencies across the observed history, but temporal attention in patch-summary construction and intra-patch processing reduces to single-token operations. Short-term variations and longer-range dependencies therefore rely on the same patch-level temporal attention.

Patch length P	Patches / window	In-domain RMSE		OOS RMSE	
		Raw	Normalized	Raw	Normalized
1	200	0.2669	0.22673	1.1814	0.22043
5	40	**0.2278**	0.20818	1.0674	**0.20794**
10	20	0.2281	**0.20760**	**1.0548**	0.20802
20	10	0.2317	0.20948	1.0660	0.20961
40	5	0.2329	0.20762	1.0825	0.21087

Table 12: Patch-length sensitivity of FilterFM. In-domain values are trajectory-level means over the 280 test trajectories from the 14 main benchmark systems; OOS values are trajectory-level means over 8,207 held-out zero-shot trajectories. All runs use a single seed, predict the $0.1/0.5/0.9$ quantiles, and use one prior-to-posterior state-reconstruction block. The shaded row is the $P = 20$ setting used in the main experiments. Lower is better.

OOS errors rise at both ends of the tested patch-length range. Our main model uses $P = 20$.

D Additional Method Details

We detail the implementation of our model, FilterFM, including observation encoding, prototype-based variable interaction, prior-to-posterior state reconstruction, and the training objective. We follow the notation in Section 4, with hidden width d, and omit block indices, projection biases, dropout, and pre-layer normalization where they do not affect the explanation.

D.1 Representing Individual Observations

The encoder begins with one token for each observation channel and time step. For $t = pL_p + r$, with patch index $p \geq 0$ and within-patch position $r \in \{1, \ldots, L_p\}$, we form

$$\begin{cases} \mathbf{c}_{p,r,m} = [\tilde{y}_{t,m}; a_m; \rho_r; \tau_t] \in \mathbb{R}^4, \\ \mathbf{o}_{p,r,m} = \mathrm{ObsEmb}(\mathbf{c}_{p,r,m}) + \mathbf{e}^y_m + \mathbf{e}^{\mathrm{patch}}_p + \mathbf{e}^{\mathrm{pos}}_r \in \mathbb{R}^d. \end{cases}$$

Here $\tilde{y}_{t,m} \in \mathbb{R}$ is the normalized observation, and $a_m \in \{0, 1\}$ indicates whether observation channel m is valid for the current system. The scalars $\rho_r = (r-1)/\max(L_p - 1, 1)$ and $\tau_t = (t-1)/\max(T-1, 1)$ lie in $[0, 1]$ and encode relative positions within the patch and the full window. The shared MLP $\mathrm{ObsEmb} : \mathbb{R}^4 \to \mathbb{R}^d$ embeds these features. The learned vectors $\mathbf{e}^y_m, \mathbf{e}^{\mathrm{patch}}_p, \mathbf{e}^{\mathrm{pos}}_r \in \mathbb{R}^d$ encode observation-channel identity, patch index, and within-patch position, respectively.

D.2 Summarizing Completed Patches

Before forming patch summaries, causal temporal encoding processes each channel's observation tokens within the patch. Let J_{enc} be the number of patch-summary temporal encoding layers and set $\mathbf{o}^{(0)}_{p,r,m} = \mathbf{o}_{p,r,m}$. For $j = 1, \ldots, J_{\mathrm{enc}}$, the updates and final encoded token are

$$\begin{cases} \mathbf{h}^{(j)}_{p,r,m} = \mathbf{o}^{(j-1)}_{p,r,m} + \mathrm{SelfAttn}_j(\mathbf{o}^{(j-1)}_{p,1:r,m}), \\ \mathbf{o}^{(j)}_{p,r,m} = \mathbf{h}^{(j)}_{p,r,m} + \mathrm{FFN}_j(\mathbf{h}^{(j)}_{p,r,m}), \\ \tilde{\mathbf{o}}_{p,r,m} = \mathbf{o}^{(J_{\mathrm{enc}})}_{p,r,m} \in \mathbb{R}^d. \end{cases}$$

Here $\mathbf{h}^{(j)}_{p,r,m}, \mathbf{o}^{(j)}_{p,r,m} \in \mathbb{R}^d$, and attention is causal within each patch. Our main model uses $J_{\mathrm{enc}} = 2$.

We then combine three components into the completed-patch summary $\mathbf{s}_{p,m}$:

$$\begin{cases} \mathbf{s}^{\mathrm{raw}}_{p,m} = \mathbf{W}_{\mathrm{lin}}\,\mathrm{vec}(\mathbf{c}_{p,1:L_p,m}), \\ \pi_{p,r,m} = \mathrm{softmax}_r\left(\dfrac{\mathbf{q}_{\mathrm{pool}}^{\top}\mathbf{W}_{\mathrm{pool,k}}\tilde{\mathbf{o}}_{p,r,m}}{\sqrt{d}}\right), \\ \mathbf{s}^{\mathrm{pool}}_{p,m} = \mathrm{AttentivePool}(\tilde{\mathbf{o}}_{p,1:L_p,m}) = \displaystyle\sum_{r=1}^{L_p}\pi_{p,r,m}\mathbf{W}_{\mathrm{pool,v}}\tilde{\mathbf{o}}_{p,r,m}, \\ \mathbf{s}^{\mathrm{last}}_{p,m} = \mathbf{W}_{\mathrm{last}}\tilde{\mathbf{o}}_{p,L_p,m}, \\ \mathbf{s}_{p,m} = \mathrm{LayerNorm}\big(\mathbf{s}^{\mathrm{raw}}_{p,m} + \mathbf{s}^{\mathrm{pool}}_{p,m} + \mathbf{s}^{\mathrm{last}}_{p,m}\big). \end{cases}$$

The operator vec concatenates the four-dimensional vectors $\mathbf{c}_{p,1,m},\ldots,\mathbf{c}_{p,L_p,m}$ in temporal order into a vector in $\mathbb{R}^{4L_p}$. The learned projections $\mathbf{W}_{\mathrm{lin}} \in \mathbb{R}^{d\times 4L_p}$ and $\mathbf{W}_{\mathrm{last}} \in \mathbb{R}^{d\times d}$ contain $4L_pd$ and d^2 weights, respectively, each with an additional d-dimensional bias omitted above. At $L_p = 20$ and $d = 256$, these are 20,736 and 65,792 parameters including biases.

Attentive pooling uses a learned query $\mathbf{q}_{\mathrm{pool}} \in \mathbb{R}^d$ and learned key and value projections $\mathbf{W}_{\mathrm{pool,k}}, \mathbf{W}_{\mathrm{pool,v}} \in \mathbb{R}^{d\times d}$. Its scalar weights $\pi_{p,r,m} \in [0,1]$ sum to one over the L_p positions. Thus, $\mathbf{s}^{\mathrm{raw}}_{p,m}$ preserves position-specific raw features through concatenation, $\mathbf{s}^{\mathrm{pool}}_{p,m}$ weights encoded positions by their learned relevance, and $\mathbf{s}^{\mathrm{last}}_{p,m}$ projects the final token, whose causal encoding covers the completed patch. All three vectors and their normalized sum $\mathbf{s}_{p,m}$ belong to $\mathbb{R}^d$.

D.3 Temporal and Variable Interaction

The encoder combines causal temporal interaction with prototype-based variable interaction at both levels. Following Section 4.1, $\ell \in \{\mathrm{patch}, \mathrm{intra}\}$ denotes the encoding level, with $s = p$ for patches and $s = (p, r)$ for intra-patch positions. Block indices are omitted, and the symbols refer to the block under consideration, with independently learned parameters across blocks. For channel m, the column vectors $\mathbf{u}^{\ell}_{s,m}, \bar{\mathbf{u}}^{\ell}_{s,m}, \mathbf{u}^{\ell,\mathrm{out}}_{s,m} \in \mathbb{R}^d$ denote its input, temporal output, and full block output, respectively. Their transposes form the m-th rows of $\mathbf{U}^{\ell}_s, \bar{\mathbf{U}}^{\ell}_s, \mathbf{U}^{\ell,\mathrm{out}}_s \in \mathbb{R}^{M\times d}$. Temporal interaction follows Eq. (3): patch-level attention accesses summaries up to patch p, while intra-patch attention accesses positions up to r within the current patch.

Prototype-based variable interaction exchanges information across observation channels through C shared slots. Its input $\mathbf{v}^{\ell}_{s,m} \in \mathbb{R}^d$ is the representation of observation channel m in $\mathbf{V}^{\ell}_s \in \mathbb{R}^{M\times d}$. At the patch level, $\mathbf{V}^{\mathrm{patch}}_p = \bar{\mathbf{U}}^{\mathrm{patch}}_p$. At the intra-patch level, the temporal output first incorporates historical context through the cross-scale conditioning described in Appendix D.4. The learned matrices $\mathbf{A}^{\ell}_q, \mathbf{A}^{\ell}_v \in \mathbb{R}^{d\times d}$ project these representations to queries and values, while the rows $\mathbf{k}^{\ell,\pm}_c \in \mathbb{R}^d$ of $\mathbf{K}^{\ell}_{\pm} \in \mathbb{R}^{C\times d}$ provide positive and negative attention keys:

$$\begin{cases} \alpha^{\ell,\pm}_{s,m,c} = \mathrm{softmax}_c\left(\dfrac{(\mathbf{A}^{\ell}_q\mathbf{v}^{\ell}_{s,m})^{\top}\mathbf{k}^{\ell,\pm}_c}{\sqrt{d}}\right), \\ \mathbf{p}^{\ell}_{s,c} = \dfrac{1}{\sum_{m'=1}^{M} a_{m'}}\displaystyle\sum_{m'=1}^{M} a_{m'}\big(\alpha^{\ell,+}_{s,m',c} - \lambda_\ell\alpha^{\ell,-}_{s,m',c}\big)\mathbf{A}^{\ell}_v\mathbf{v}^{\ell}_{s,m'}. \end{cases}$$

The query and value projections $\mathbf{A}^{\ell}_q$ and $\mathbf{A}^{\ell}_v$ are learned independently in each block and are separate from the temporal-attention, prototype-reassembly, and Retrieve projections. Within each block, positive and negative attention share these query and value projections but use distinct keys $\mathbf{K}^{\ell}_+$ and $\mathbf{K}^{\ell}_-$. The scalar weights $\alpha^{\ell,+}_{s,m,c}, \alpha^{\ell,-}_{s,m,c} \in [0,1]$ each sum to one over c, and $\lambda_\ell \in \mathbb{R}$ controls the contribution of negative attention, with $\lambda_\ell = 0.5$ in our main model. The keys $\mathbf{K}^{\ell}_{\pm}$ are learned parameters shared across input sequences; the prototypes $\mathbf{p}^{\ell}_{s,c} \in \mathbb{R}^d$ are computed from the current representations $\mathbf{v}^{\ell}_{s,m}$. The indicators a_m exclude invalid channels, and division by $\sum_m a_m$ averages over valid channels. When all M channels are valid, this is exactly the aggregation in Eq. (4).

After obtaining $\mathbf{p}_{s,c}^{\ell}$, we stack the C vectors as rows of $\mathbf{P}_s^{\ell} \in \mathbb{R}^{C\times d}$. Self-attention across these prototypes, followed by a residual FFN, gives

$$\begin{cases} \bar{\mathbf{P}}_s^{\ell} = \mathbf{P}_s^{\ell} + \mathrm{SelfAttn}_{\ell}(\mathbf{P}_s^{\ell}), \\ \tilde{\mathbf{P}}_s^{\ell} = \bar{\mathbf{P}}_s^{\ell} + \mathrm{FFN}_{\ell}(\bar{\mathbf{P}}_s^{\ell}), \end{cases}$$

where $\bar{\mathbf{P}}_s^{\ell}, \tilde{\mathbf{P}}_s^{\ell} \in \mathbb{R}^{C\times d}$. These operations mix prototypes at the same temporal position s. Writing the rows of $\tilde{\mathbf{P}}_s^{\ell}$ as $\tilde{\mathbf{p}}_{s,c}^{\ell} \in \mathbb{R}^d$, gated reassembly in Eq. (5) updates each variable as

$$\begin{cases} \beta_{s,m,c}^{\ell} = \mathrm{softmax}_c\left(\dfrac{(\mathbf{W}_r^{\ell,\mathrm{proto}}\mathbf{v}_{s,m}^{\ell})^{\top}(\mathbf{W}_i^{\ell,\mathrm{proto}}\tilde{\mathbf{p}}_{s,c}^{\ell})}{\sqrt{d}}\right), \\ \mathbf{r}_{s,m}^{\ell} = \displaystyle\sum_{c=1}^{C} \beta_{s,m,c}^{\ell}\mathbf{W}_s^{\ell,\mathrm{proto}}\tilde{\mathbf{p}}_{s,c}^{\ell}, \\ \mathbf{g}_{s,m}^{\ell} = \sigma\big(\mathrm{MLP}_{\ell}([\mathbf{v}_{s,m}^{\ell}; \mathbf{r}_{s,m}^{\ell}])\big), \\ \widehat{\mathbf{v}}_{s,m}^{\ell} = \mathbf{v}_{s,m}^{\ell} + \mathbf{g}_{s,m}^{\ell} \odot \mathbf{W}_o^{\ell,\mathrm{proto}}\mathbf{r}_{s,m}^{\ell}, \\ \mathbf{u}_{s,m}^{\ell,\mathrm{out}} = \widehat{\mathbf{v}}_{s,m}^{\ell} + \mathrm{FFN}_{\ell}(\widehat{\mathbf{v}}_{s,m}^{\ell}). \end{cases}$$

The learned projections $\mathbf{W}_r^{\ell,\mathrm{proto}}, \mathbf{W}_i^{\ell,\mathrm{proto}}, \mathbf{W}_s^{\ell,\mathrm{proto}}, \mathbf{W}_o^{\ell,\mathrm{proto}} \in \mathbb{R}^{d\times d}$ belong to the prototype-reassembly router, with independent parameters in each block. For each variable, $\beta_{s,m,c}^{\ell} \in [0,1]$ normalizes over C prototypes to form $\mathbf{r}_{s,m}^{\ell} \in \mathbb{R}^d$, and $\mathbf{g}_{s,m}^{\ell} \in (0,1)^d$ gates its contribution to $\widehat{\mathbf{v}}_{s,m}^{\ell} \in \mathbb{R}^d$. Stacking these retrieved vectors, gates, and updated representations as rows recovers $\mathbf{R}_s^{\ell}, \mathbf{G}_s^{\ell}, \widehat{\mathbf{V}}_s^{\ell}$ in Eq. (5).

D.4 Cross-Scale Conditioning and Local Memory

Patch-level temporal and variable interaction transform $\mathbf{s}_{p,m}$ into $\mathbf{H}_{p,m}^{\mathrm{raw}} \in \mathbb{R}^d$. The intra-patch encoder receives $\mathbf{H}_{p,m} = \mathbf{H}_{p-1,m}^{\mathrm{raw}}$ for $p \geq 1$, initialized with the learned vector $\mathbf{H}_{0,m} = \mathbf{h}_m^{\mathrm{bos}} \in \mathbb{R}^d$. *This one-patch delay prevents future leakage: time steps within patch p cannot access its completed summary, which includes observations later in that patch.*

After intra-patch temporal interaction, each variable representation reads the delayed patch memory. For a query $\mathbf{q} \in \mathbb{R}^d$ and memory $\mathbf{B} \in \mathbb{R}^{M\times d}$, whose m-th row is written as the column vector $\mathbf{B}_m \in \mathbb{R}^d$, the retrieval operator in Eq. (6) is

$$\begin{cases} \gamma_m = \mathrm{softmax}_{m:a_m=1}\left(\dfrac{(\mathbf{W}_r^{\mathrm{ret}}\mathbf{q})^{\top}(\mathbf{W}_i^{\mathrm{ret}}\mathbf{B}_m)}{\sqrt{d}}\right), \\ \mathbf{r} = \displaystyle\sum_{m:a_m=1} \gamma_m \mathbf{W}_s^{\mathrm{ret}}\mathbf{B}_m, \\ \mathbf{g} = \sigma(\mathrm{MLP}([\mathbf{q}; \mathbf{r}])), \\ \tilde{\mathbf{q}} = \mathbf{q} + \mathbf{g} \odot \mathbf{W}_o^{\mathrm{ret}}\mathbf{r}, \\ \mathrm{Retrieve}(\mathbf{q}, \mathbf{B}) = \tilde{\mathbf{q}} + \mathrm{FFN}(\tilde{\mathbf{q}}). \end{cases}$$

The learned matrices $\mathbf{W}_r^{\mathrm{ret}}, \mathbf{W}_i^{\mathrm{ret}}, \mathbf{W}_s^{\mathrm{ret}}, \mathbf{W}_o^{\mathrm{ret}} \in \mathbb{R}^{d\times d}$ project the query, memory keys, memory values, and retrieved vector, respectively. These parameters are independent of the prototype-reassembly parameters in Appendix D.3. The scalar weights $\gamma_m \in [0,1]$ normalize only over valid channels with $a_m = 1$. Both the retrieved vector $\mathbf{r}$ and updated query $\tilde{\mathbf{q}}$ lie in $\mathbb{R}^d$. The gate MLP maps $[\mathbf{q}; \mathbf{r}] \in \mathbb{R}^{2d}$ to $\mathbf{g} \in (0,1)^d$, controlling the memory contribution to each query feature.

For cross-scale conditioning, the query is the temporal output $\bar{\mathbf{u}}_{p,r,m}^{\mathrm{intra}}$, and the memory contains the delayed summaries $\mathbf{H}_{p,1:M}$. This produces the same history-conditioned variable representation as Eq. (7):

$$\mathbf{v}_{p,r,m}^{\mathrm{intra}} = \mathrm{Retrieve}\big(\bar{\mathbf{u}}_{p,r,m}^{\mathrm{intra}}, \mathbf{H}_{p,1:M}\big) \in \mathbb{R}^d.$$

Stacking $\mathbf{v}_{p,r,m}^{\mathrm{intra}}$ over channels gives $\mathbf{V}_{p,r}^{\mathrm{intra}} \in \mathbb{R}^{M\times d}$, which undergoes prototype-based variable interaction through Eqs. (4)–(5). Each subsequent intra-patch block uses the preceding block's

$\mathbf{U}_{p,r}^{\text{intra,out}}$ as its input $\mathbf{U}_{p,r}^{\text{intra}}$ and repeats temporal interaction, history retrieval, and variable interaction. For the final block, $\mathbf{L}_{t,m} = \mathbf{u}_{(p,r),m}^{\text{intra,out}} \in \mathbb{R}^d$ is the local memory used for posterior reconstruction; its one-step shift $\mathbf{L}_{t,m}^- = \mathbf{L}_{t-1,m}$, with $\mathbf{L}_{0,m} = \mathbf{0}$, is used for prior prediction.

D.5 State Queries and State Reconstruction

At each time step, the encoder provides M observation-channel memory vectors $\mathbf{L}_{t,m}$. To estimate the N latent state components, the reconstruction module assigns a learned query to each state channel n and uses it to retrieve information from all valid observation channels. This permits N and M to differ. For $t = pL_p + r$, the initial query is

$$\mathbf{q}_{t,n} = \mathbf{e}_n^x + \mathbf{e}_p^{\text{patch}} + \mathbf{e}_r^{\text{pos}} \in \mathbb{R}^d.$$

Here $\mathbf{e}_n^x \in \mathbb{R}^d$ is the sum of the learned embedding for state channel n and a learned vector shared by all state queries. The patch-index embedding $\mathbf{e}_p^{\text{patch}}$ and within-patch-position embedding $\mathbf{e}_r^{\text{pos}}$ are shared with the observation encoder. The query $\mathbf{q}_{t,n}$ contains no observation or target state value; it obtains evidence through the following two retrieval operations.

For prior prediction, $\mathbf{q}_{t,n}$ queries shifted memory $\mathbf{L}_{t,1:M}^- = \mathbf{L}_{t-1,1:M}$ to form the prior latent representation $\mathbf{z}_{t,n}^- \in \mathbb{R}^d$ from observations before t. This prior then queries current memory $\mathbf{L}_{t,1:M}$ to obtain current evidence $\mathbf{e}_{t,n} \in \mathbb{R}^d$ for the posterior update at t:

$$\begin{cases} \mathbf{z}_{t,n}^- = \text{Retrieve}(\mathbf{q}_{t,n}, \mathbf{L}_{t,1:M}^-), \\ \mathbf{e}_{t,n} = \text{Retrieve}(\mathbf{z}_{t,n}^-, \mathbf{L}_{t,1:M}). \end{cases}$$

The two retrieval operations use Eq. (6) with separate projections, gates, and FFNs, independent of those used for cross-scale conditioning in Appendix D.4. The posterior update then concatenates the prior, current evidence, and their difference into a vector in $\mathbb{R}^{3d}$, as described in Section 4.2:

$$\begin{cases} \mathbf{g}_{t,n} = \sigma\big(\text{MLP}_{\text{gate}}([\mathbf{z}_{t,n}^-; \mathbf{e}_{t,n}; \mathbf{e}_{t,n} - \mathbf{z}_{t,n}^-])\big), \\ \mathbf{\Delta}_{t,n} = \text{MLP}_{\text{update}}([\mathbf{z}_{t,n}^-; \mathbf{e}_{t,n}; \mathbf{e}_{t,n} - \mathbf{z}_{t,n}^-]), \\ \mathbf{z}_{t,n}^+ = \mathbf{z}_{t,n}^- + \mathbf{g}_{t,n} \odot \mathbf{\Delta}_{t,n}. \end{cases}$$

The difference $\mathbf{e}_{t,n} - \mathbf{z}_{t,n}^-$ compares current evidence with the prior in the learned feature space. The two MLPs map $\mathbb{R}^{3d}$ to $\mathbb{R}^d$, producing the sigmoid gate $\mathbf{g}_{t,n} \in (0,1)^d$ and correction $\mathbf{\Delta}_{t,n} \in \mathbb{R}^d$ that form the posterior latent representation $\mathbf{z}_{t,n}^+ \in \mathbb{R}^d$. Learned quantile projections $W_\tau^+ \in \mathbb{R}^{1\times d}$ yield scalar normalized state quantiles, with the median mapped back to the original state scale:

$$\begin{cases} Q_{t,n,\tau}^+ = W_\tau^+ \mathbf{z}_{t,n}^+ \in \mathbb{R}, \quad \tau \in \mathcal{Q} = \{0.1, 0.5, 0.9\}, \\ \hat{x}_{t,n} = \sigma_n^x Q_{t,n,0.5}^+ + \mu_n^x \in \mathbb{R}. \end{cases}$$

D.6 Exact Training Loss

Training supervises the state quantiles in the normalized coordinates defined in Section 4.1. Let Ω_b contain the valid time–state pairs (t,n) in sample b, and let $\mathcal{Q} = \{0.1, 0.5, 0.9\}$. The posterior pinball loss is defined below, with the same form applied to prior quantiles Q^- for $\mathcal{L}_{\text{prior}}$:

$$\begin{cases} \mathcal{L}_{\text{post}} = \dfrac{1}{B}\displaystyle\sum_{b=1}^{B} \frac{1}{|\Omega_b|\,|\mathcal{Q}|} \sum_{(t,n)\in\Omega_b} \sum_{\tau\in\mathcal{Q}} \rho_\tau\Big(\tilde{x}_{b,t,n} - Q_{b,t,n,\tau}^+\Big), \\ \rho_\tau(e) = \max\{\tau e, (\tau-1)e\}. \end{cases} \tag{24}$$

The median Huber term provides an additional point-estimation objective. With transition scale $\delta > 0$, the implementation uses the smooth-L1 form

$$\begin{cases} \mathcal{L}_{\text{huber}} = \dfrac{1}{B}\displaystyle\sum_{b=1}^{B} \frac{1}{|\Omega_b|} \sum_{(t,n)\in\Omega_b} \ell_\delta\Big(\tilde{x}_{b,t,n} - Q_{b,t,n,0.5}^+\Big), \\ \ell_\delta(e) = \begin{cases} e^2/(2\delta), & |e| < \delta, \\ |e| - \delta/2, & |e| \geq \delta. \end{cases} \end{cases}$$

This term is quadratic for small errors and linear for large errors, retaining sensitivity to local estimation accuracy without the quadratic growth of MSE on large residuals.

Prototype-key regularization acts on the learned positive and negative key matrices $\mathbf{K}_{\pm}^{\ell}$ in Eq. (4). Here $j = 1, \ldots, J$ indexes all patch-level and intra-patch blocks, whose individual indices were omitted in that equation. For block j, $\bar{\mathbf{K}}_{j,+}, \bar{\mathbf{K}}_{j,-} \in \mathbb{R}^{C \times d}$ denote its key matrices after normalizing each row to unit ℓ_2 norm. The main model has $J = 5$: three patch-level blocks and two intra-patch blocks. The regularizer is

$$\begin{cases} \mathcal{R}_j = \dfrac{1}{C^2} \left\| \bar{\mathbf{K}}_{j,+} \bar{\mathbf{K}}_{j,-}^{\top} \right\|_F^2 + \dfrac{1}{2C^2} \displaystyle\sum_{\eta \in \{+,-\}} \left\| \bar{\mathbf{K}}_{j,\eta} \bar{\mathbf{K}}_{j,\eta}^{\top} - \mathbf{I}_C \right\|_F^2, \\ \mathcal{L}_{\text{orth}} = \dfrac{1}{J} \displaystyle\sum_{j=1}^{J} \mathcal{R}_j. \end{cases}$$

Here $\| \cdot \|_F$ is the Frobenius norm. The first term penalizes alignment across the two key sets, while the remaining terms encourage distinct directions among keys within each set. The full objective combines posterior estimation, prior supervision, median accuracy, and key diversity:

$$\mathcal{L} = \mathcal{L}_{\text{post}} + \lambda_{\text{prior}} \mathcal{L}_{\text{prior}} + \lambda_{\text{huber}} \mathcal{L}_{\text{huber}} + \lambda_{\text{orth}} \mathcal{L}_{\text{orth}}.$$

The coefficients set the relative contributions of the auxiliary terms, with λ_{prior} studied explicitly in Table 7. Joint prior and posterior supervision encourages the model to learn both a useful prediction from history and its refinement after receiving the current observation.

D.7 Main Model Training Configuration

Table 13 lists the hyperparameters used to train the main FilterFM model on the 15,120 trajectories described in Appendix B.3. Training starts from random initialization, and the checkpoint with the lowest validation raw RMSE is used for both in-domain and zero-shot evaluation.

Hyperparameter	Setting
Architecture and input representation	
Window length T	200
Patch length L_p	20
Maximum observation / state dimension	100 / 100
Hidden width d	256
Self-attention heads	4
FFN hidden width	1,024
Patch-summary temporal encoding layers J_{enc}	2
Patch-level blocks	3
Intra-patch blocks	2
Prototype self-attention layers per block	1
Prior-to-posterior state-reconstruction blocks	1
Prototype slots C	8
Negative-attention coefficient λ_ℓ	0.5
Dropout probability	0.1
Activation	GELU
Gate bias initialization	-2.0
Observation and state normalization	Per-channel instance normalization

Hyperparameter	Setting
Training objective	
Quantile levels $\mathcal{Q}$	$\{0.1, 0.5, 0.9\}$
Prior-loss weight λ_{prior}	1.0
Median Huber weight λ_{huber}	0.5
Huber transition scale δ	0.5
Prototype-key regularization weight λ_{orth}	10^{-3}
Optimization and checkpoint selection	
Optimizer	AdamW
AdamW coefficients (β_1, β_2)	$(0.9, 0.999)$
AdamW numerical stabilizer ϵ_{opt}	10^{-8}
Initial learning rate	10^{-4}
Learning-rate schedule	Cosine decay to 10^{-8} over 20 epochs
Weight decay	10^{-5}
Gradient ℓ_2-norm clipping threshold	1.0
Batch size	2 trajectories
Training epochs	20
Early-stopping patience	15 epochs
Mixed-precision training	bfloat16
Checkpoint selection criteria	Validation raw RMSE

Table 13: Architecture and training hyperparameters of the main FilterFM model.

D.8 Interpreting Patch-Level Interaction

Figure 11 shows all four temporal-attention heads in the third patch-level block of the checkpoint evaluated in Table 1. Each system uses its first test trajectory and first observation channel, with heads displayed separately rather than averaged. Each heatmap row shows attention over the available patch history, with the causal mask restricting the first row to P0. For Cubic-Sensoring100, head 1 repeatedly emphasizes P3, whereas head 2 distributes attention more broadly. In Duffing64, head 4 places more weight on intermediate patches than head 2. These patterns show how different heads access historical summaries to complement intra-patch processing of current observations.

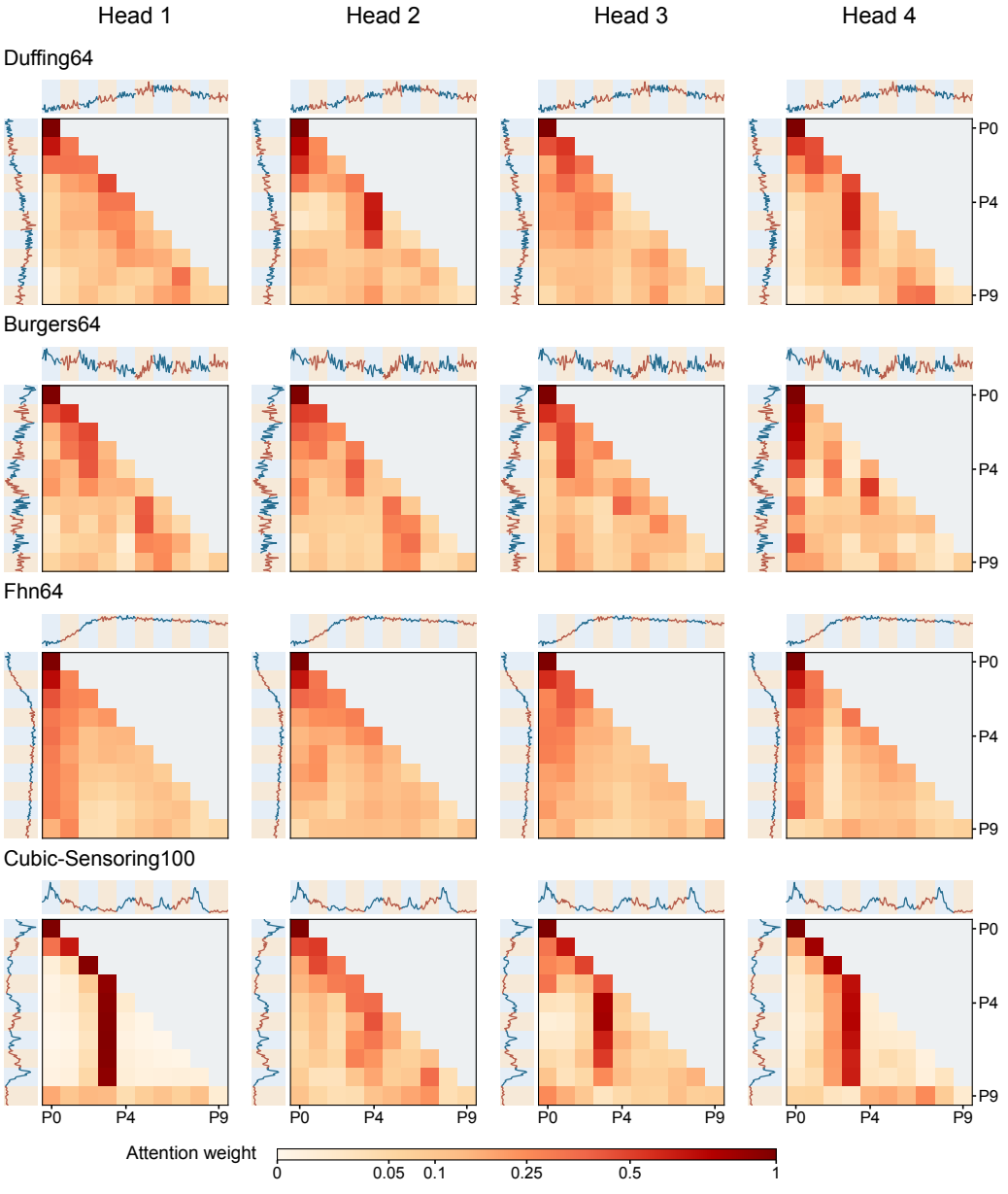

Figure 11: Patch-level temporal attention from all four heads of the third patch-level block in the Table 1 checkpoint. Each system occupies one row and heads 1–4 occupy the columns. Within each heatmap, rows are query patches and columns are key patches; gray cells are causally masked. The marginal curves show the same unsmoothed observation channel in both directions, with alternating colors marking 20-step patches. All panels share a square-root color scale over weights from 0 to 1.

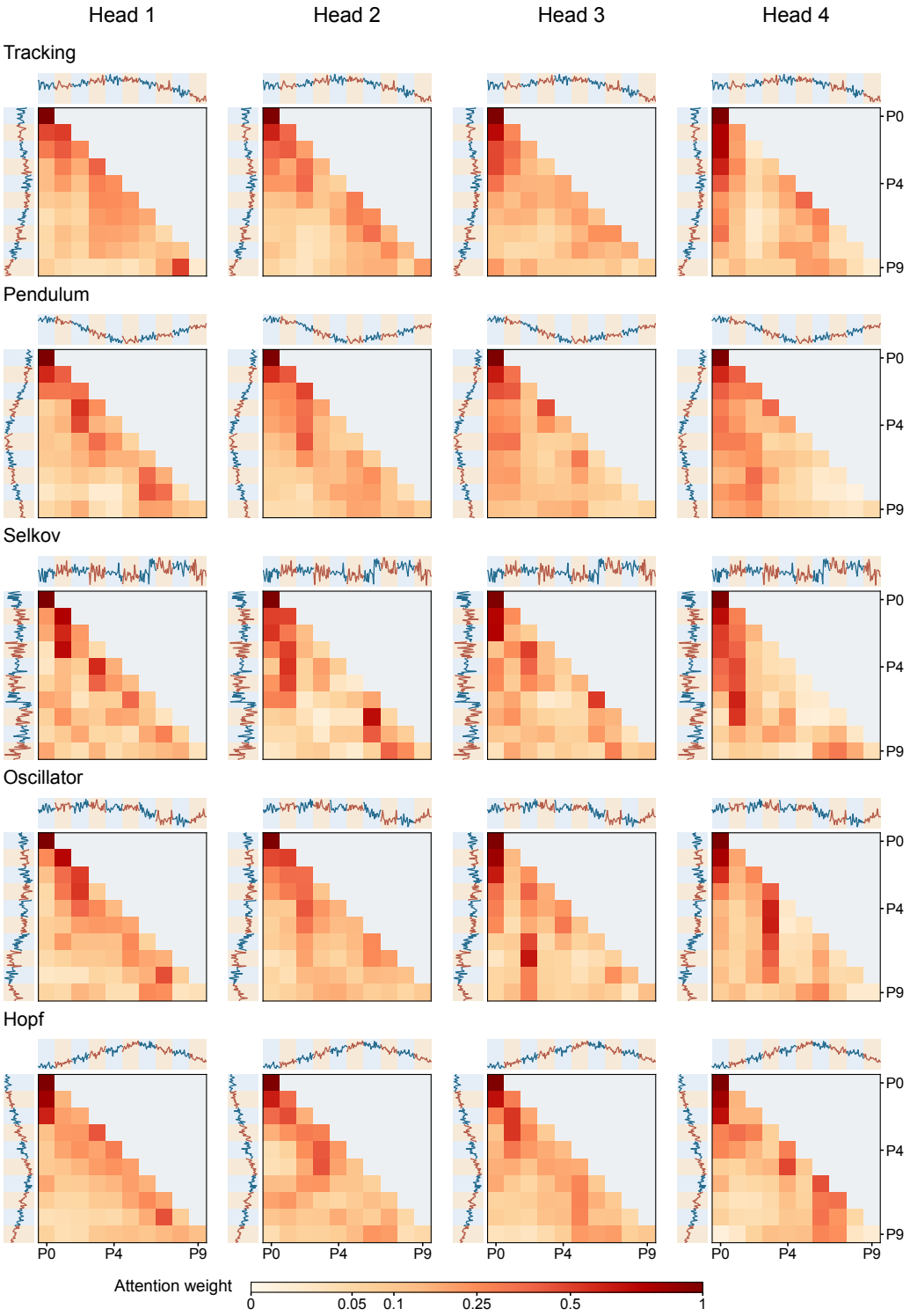

Figure 11 (continued): All four heads for Tracking, Pendulum, Selkov, Oscillator, and Hopf, using the same checkpoint, trajectory and channel selection, and color scale.

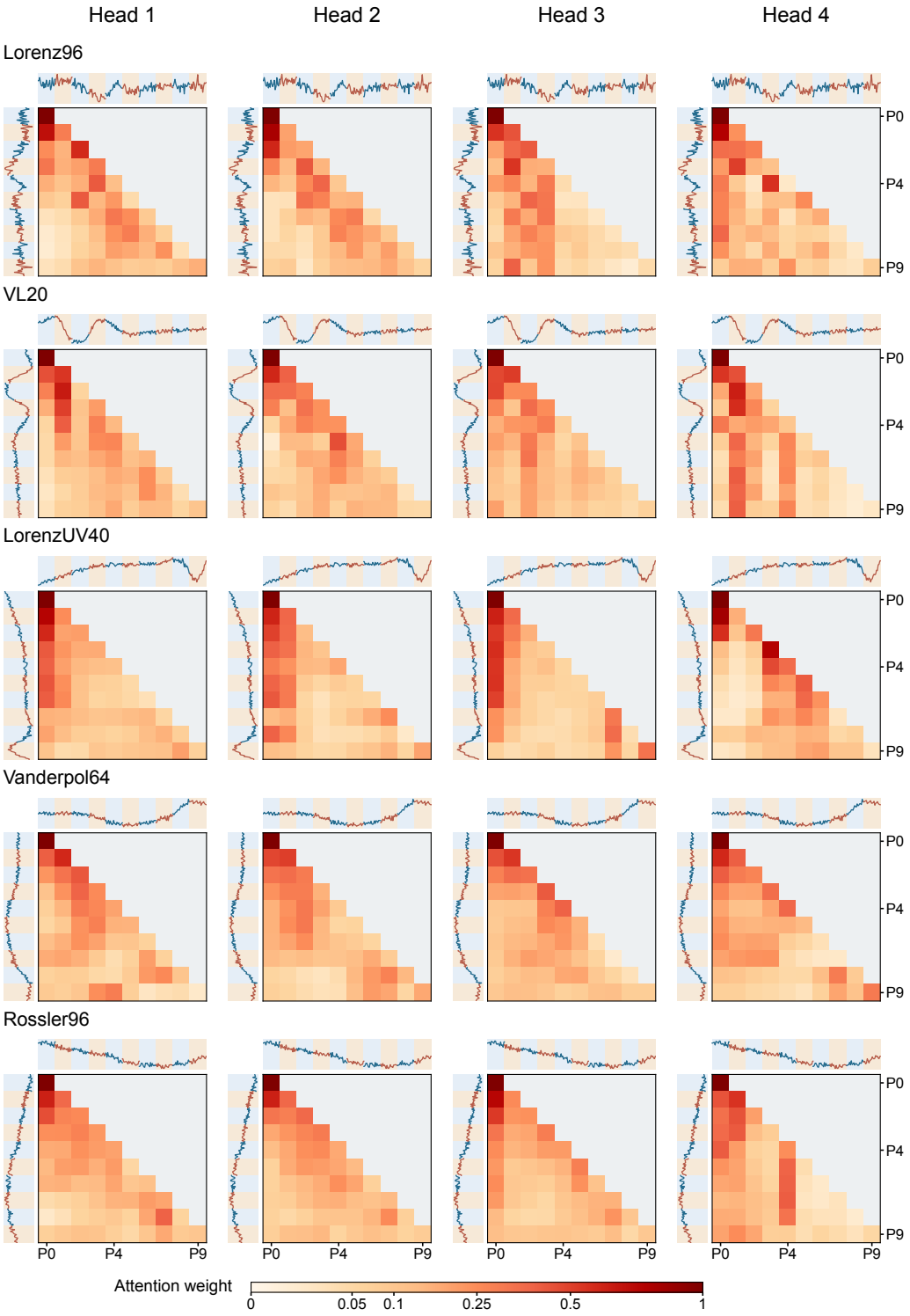

Figure 11 (continued): All four heads for Lorenz96, VL20, LorenzUV40, Vanderpol64, and Rossler96, using the same checkpoint, trajectory and channel selection, and color scale.

D.9 Interpreting Prototype-Based Variable Interaction

Figure 12 shows the coefficients used to aggregate observation-channel representations into prototypes in the third patch-level block of the Table 1 checkpoint. We use the first test trajectory of four systems and retain all observation channels.

LorenzUV40 emphasizes its first eight slow-variable channels in slot 5 at P0, then weakens these coefficients at later patches. Cubic-Sensoring100 retains a more persistent pattern of channel preferences in slot 5. These patterns show how fixed prototype keys produce input-dependent aggregation, reusing eight slots across systems with different observation dimensions.

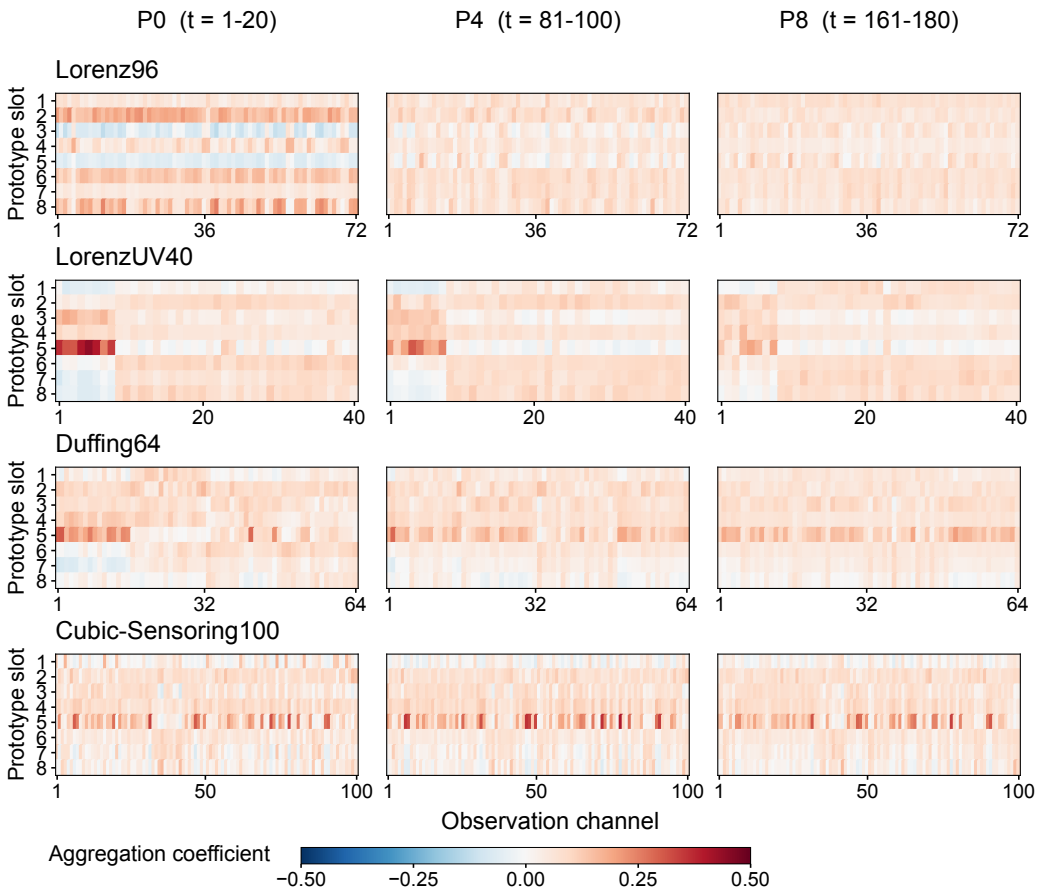

Figure 12: Coefficients $\alpha^{\ell,+}_{p,m,c} - 0.5\alpha^{\ell,-}_{p,m,c}$ for forming prototypes in Eq. (4), before the common $1/M$ factor. Rows are prototype slots and columns are observation channels. Panels share a linear color scale: red indicates positive coefficients and blue negative coefficients.

E More State Estimation Showcase

E.1 In-Domain State Estimation

Figures 13–26 compare FilterFM, LLMFilter, and FM-UKF across all fourteen benchmark systems. Each high-dimensional system is shown through six distinct state channels from three held-out trajectories; the additional channels are spread across the state-index range without using prediction errors for selection. Each row shows the same trajectory and channel for all three methods with shared axis limits, and the FilterFM panel includes its predicted 0.1–0.9 quantile interval. The low-dimensional examples retain three trajectory–channel pairs per system.

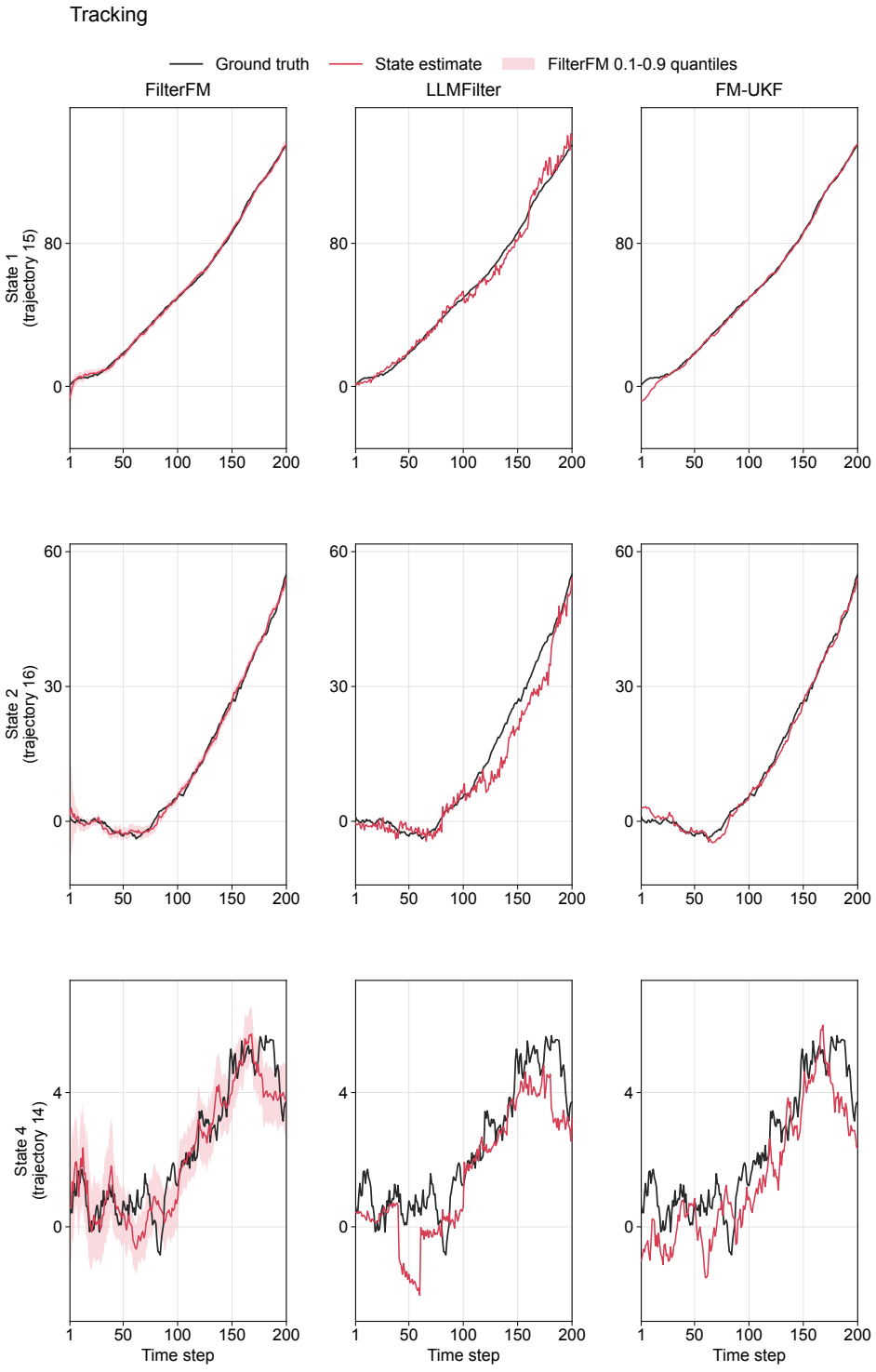

Figure 13: Additional in-domain state-estimation examples on Tracking.

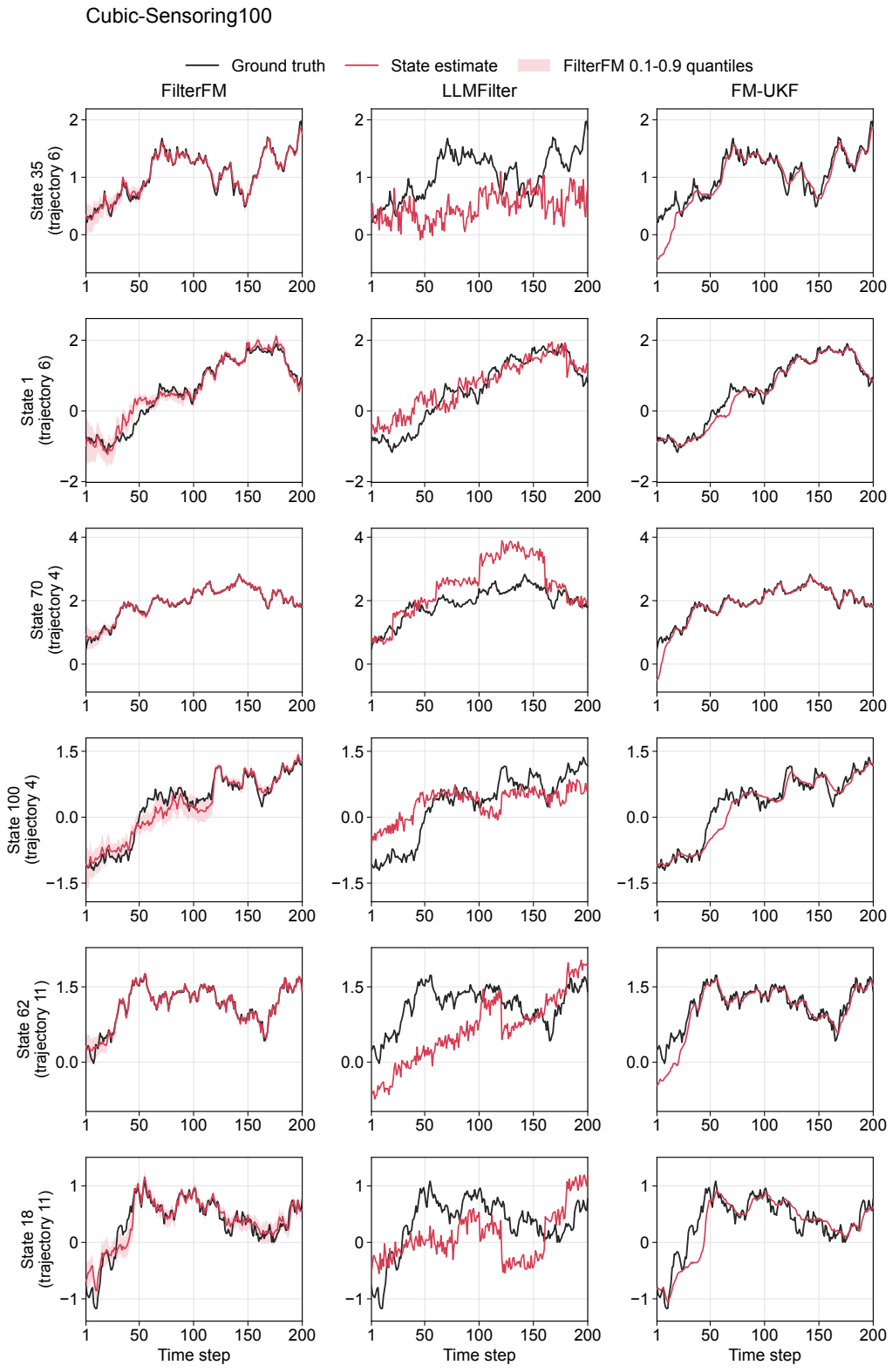

Figure 14: In-domain state estimation on Cubic-Sensoring100: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

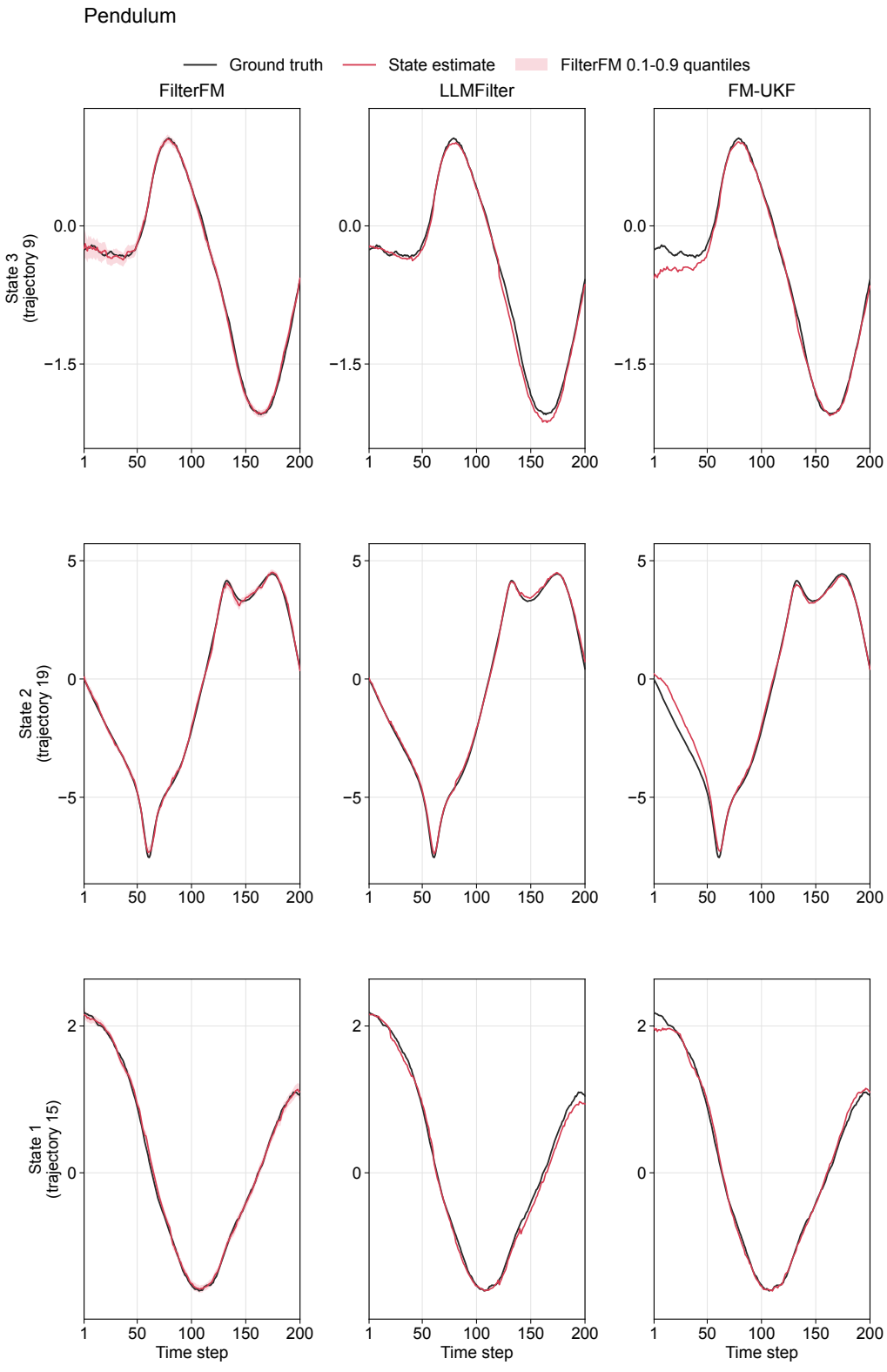

Figure 15: Additional in-domain state-estimation examples on Pendulum.

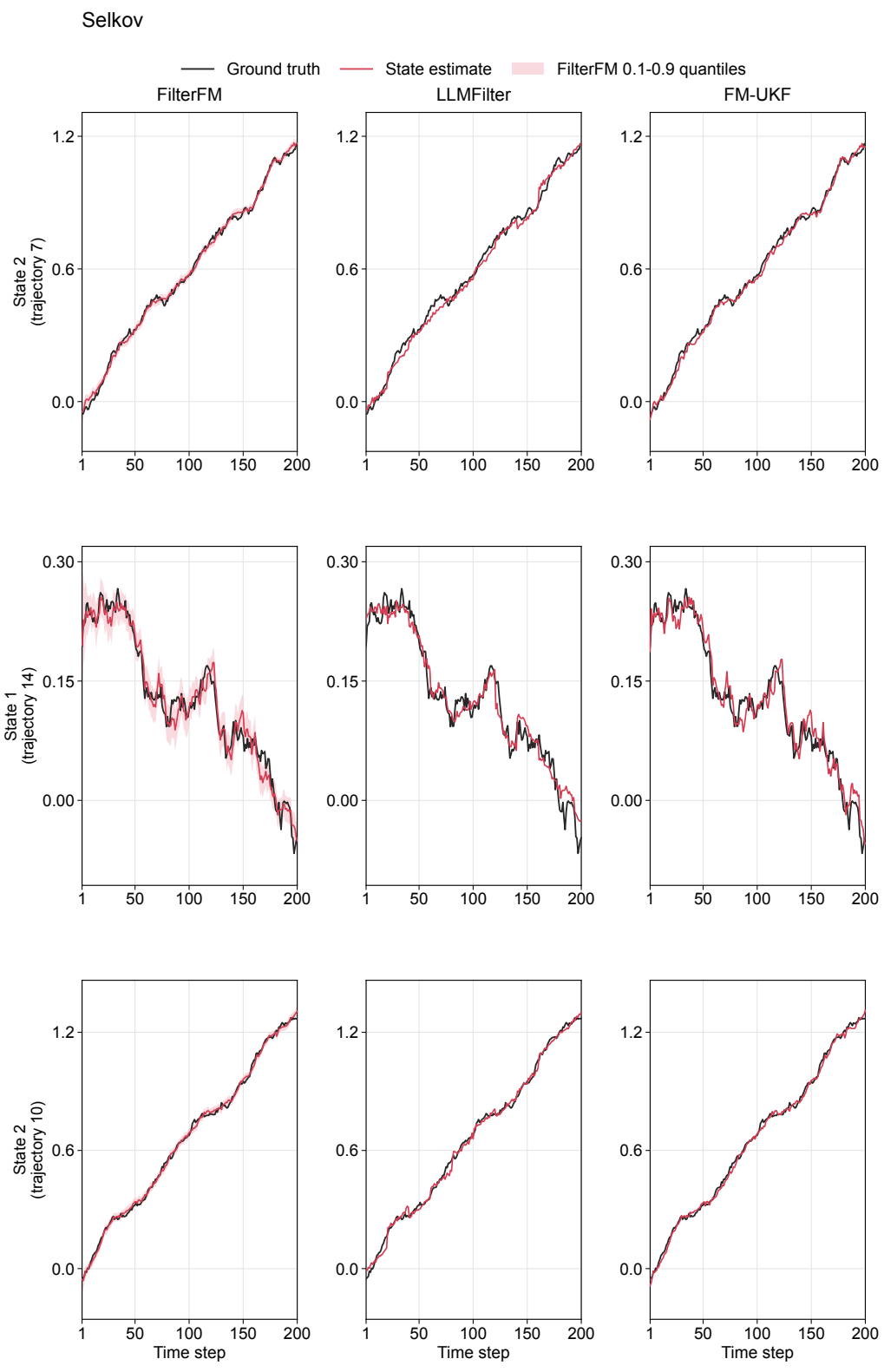

Figure 16: Additional in-domain state-estimation examples on Selkov.

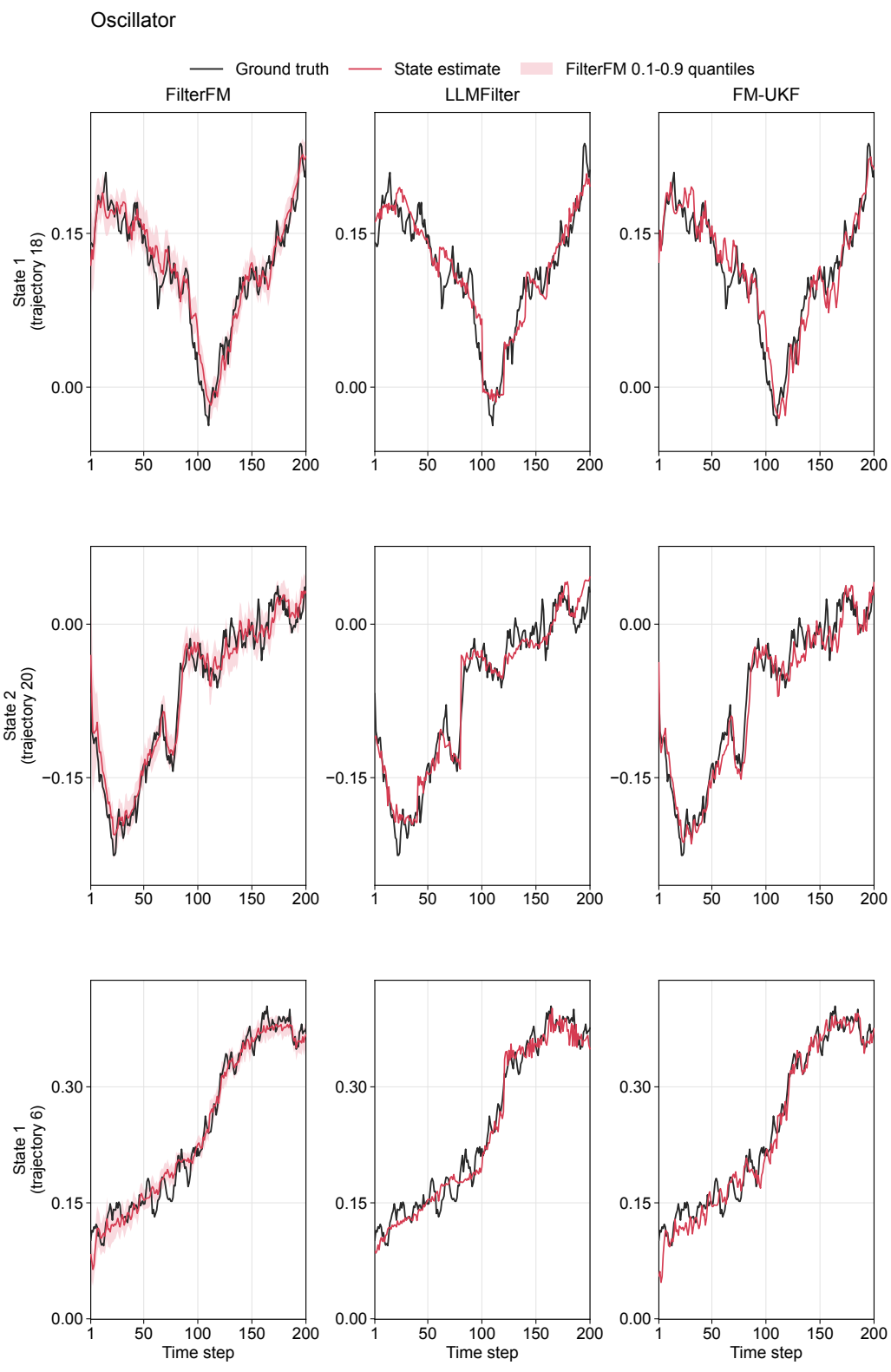

Figure 17: Additional in-domain state-estimation examples on Oscillator.

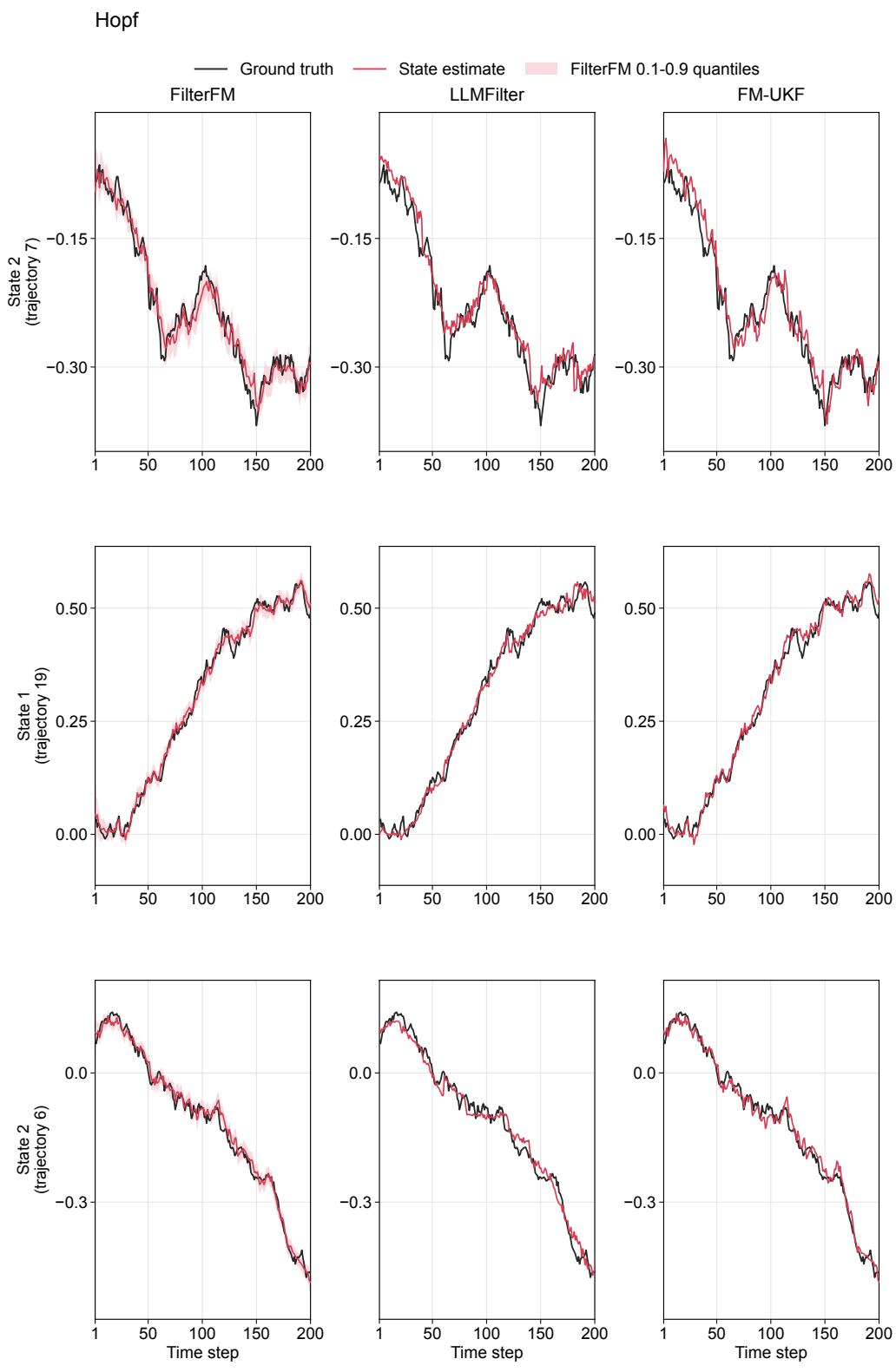

Figure 18: Additional in-domain state-estimation examples on Hopf.

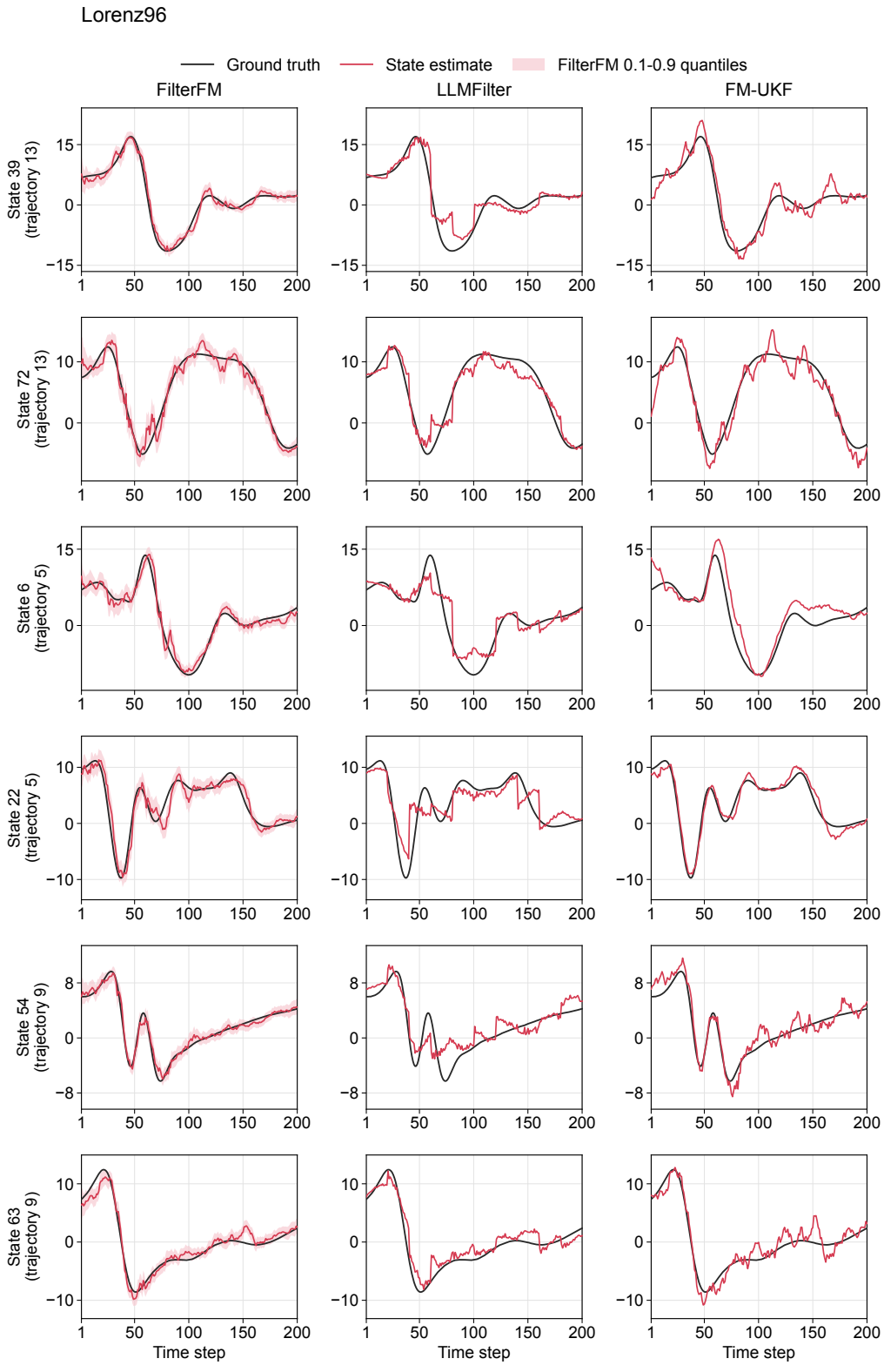

Figure 19: In-domain state estimation on Lorenz96: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

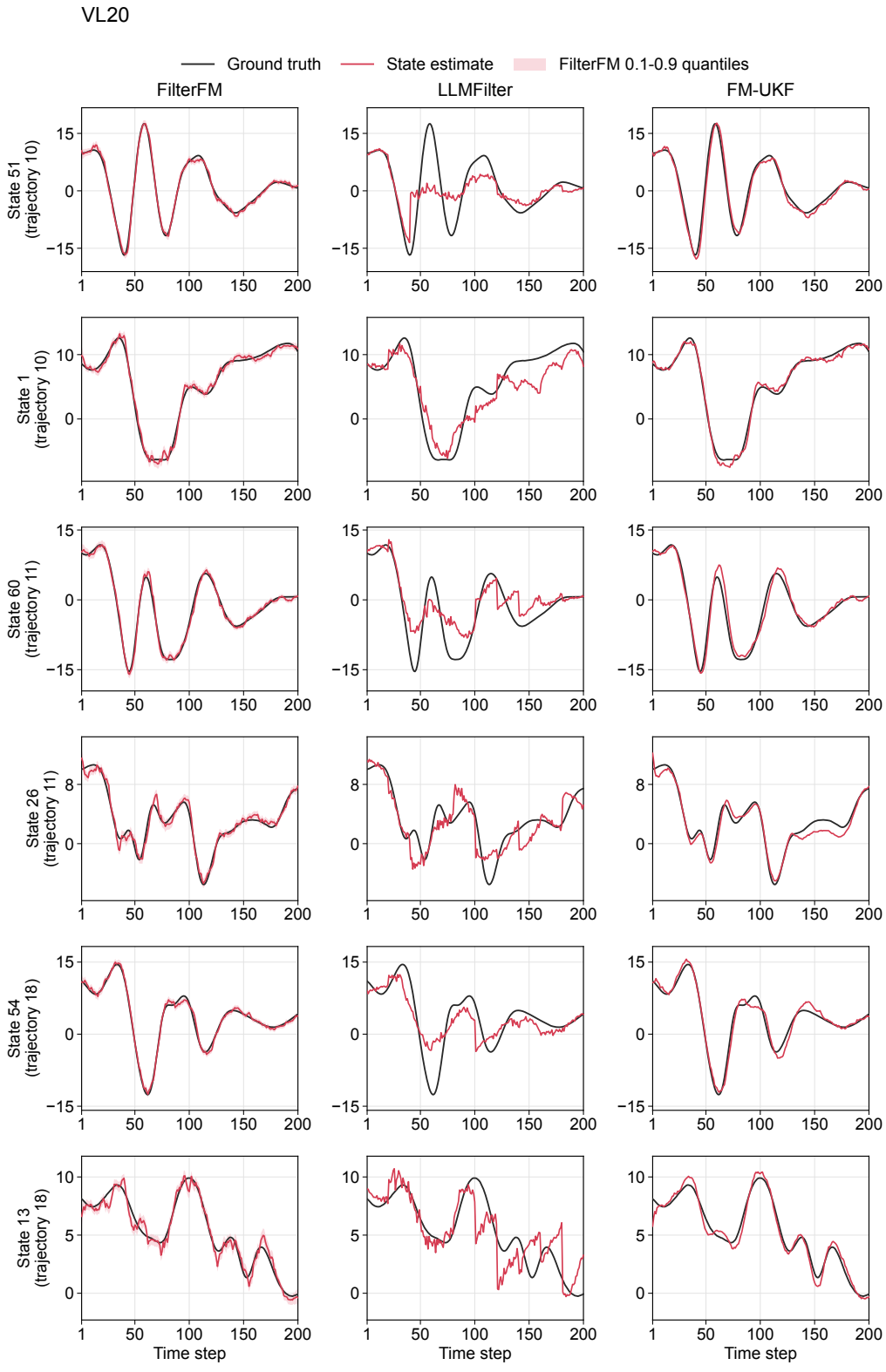

Figure 20: In-domain state estimation on VL20: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

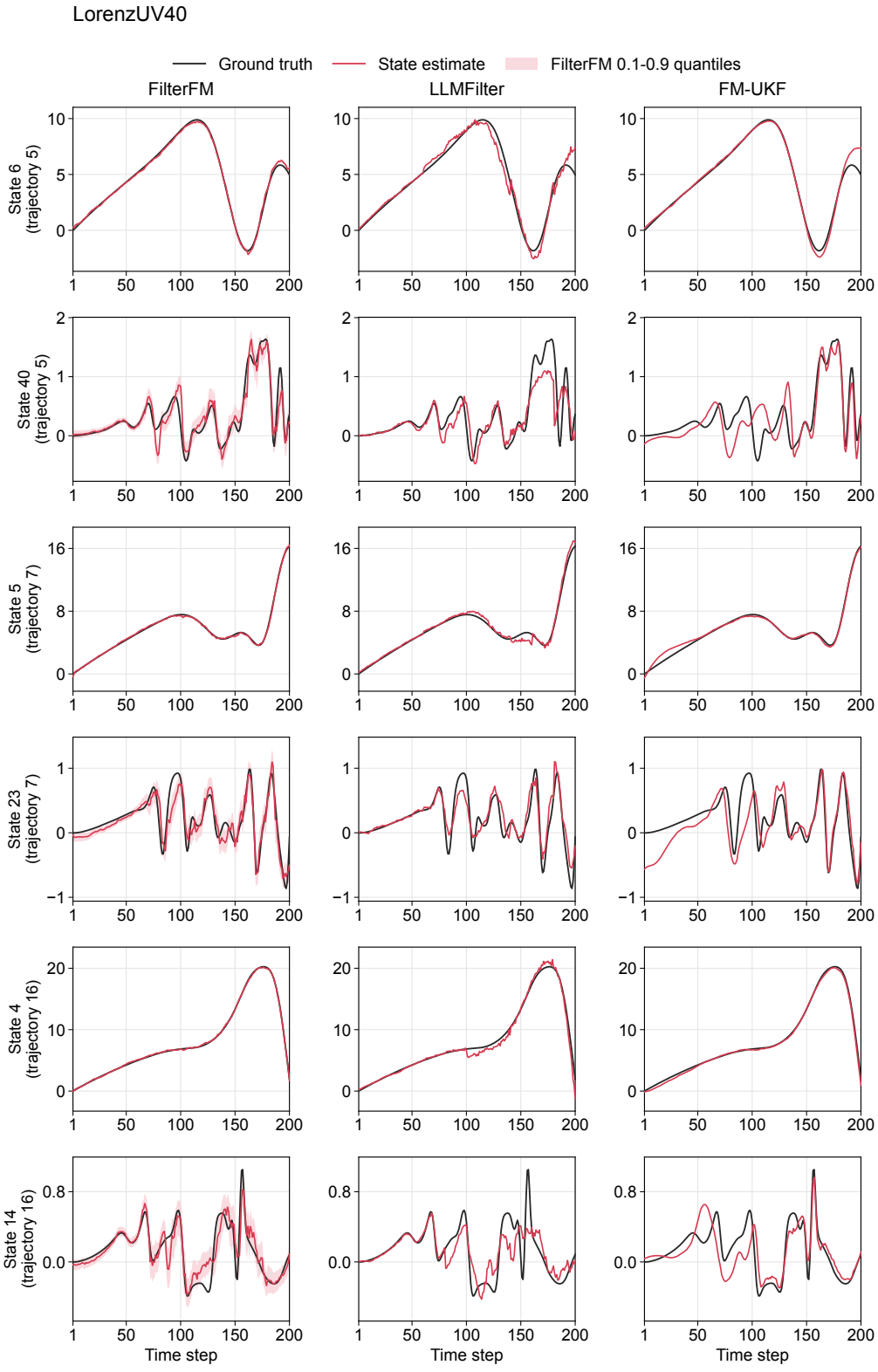

Figure 21: In-domain state estimation on LorenzUV40: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

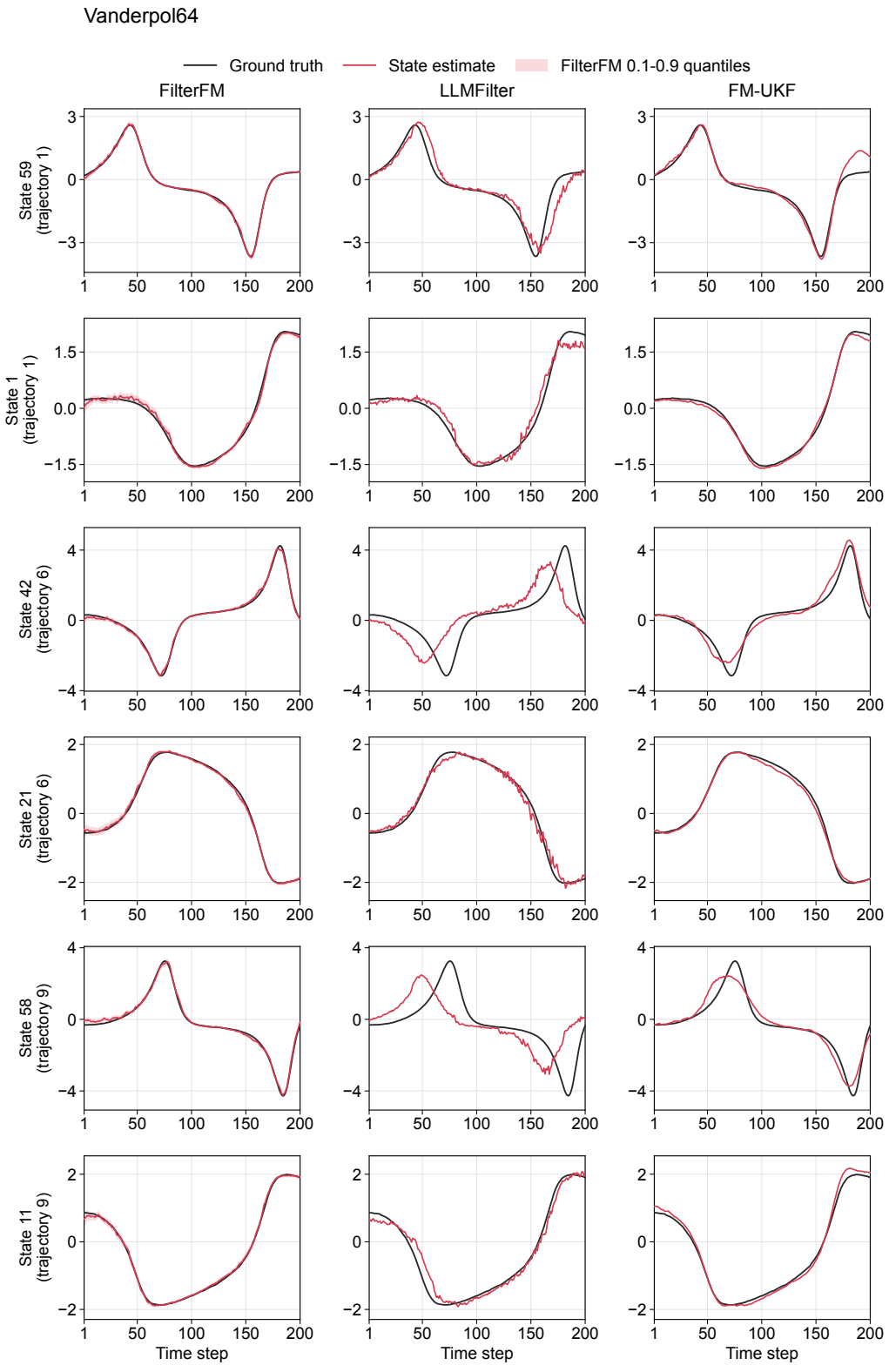

Figure 22: In-domain state estimation on Vanderpol64: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

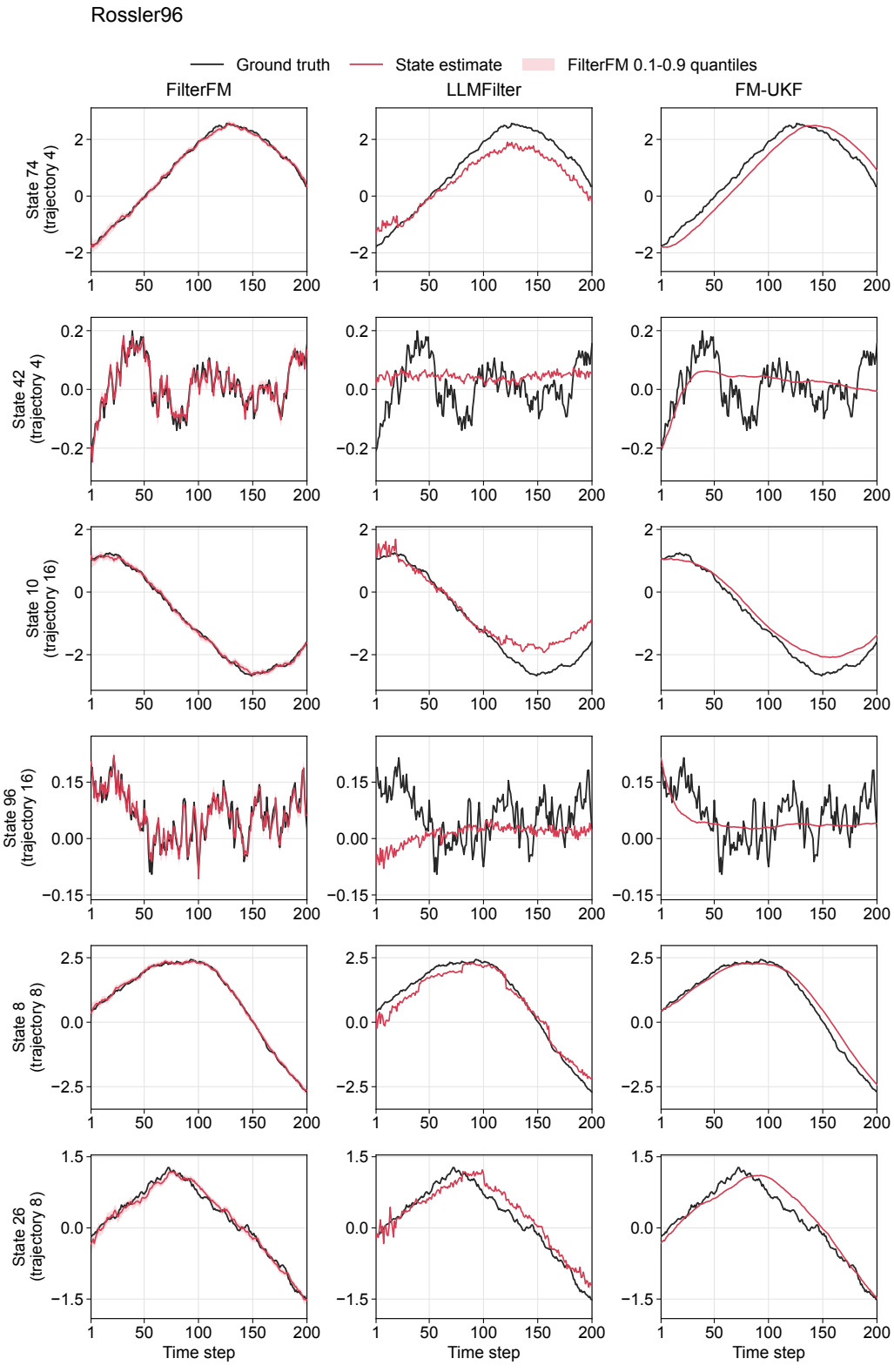

Figure 23: In-domain state estimation on Rossler96: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

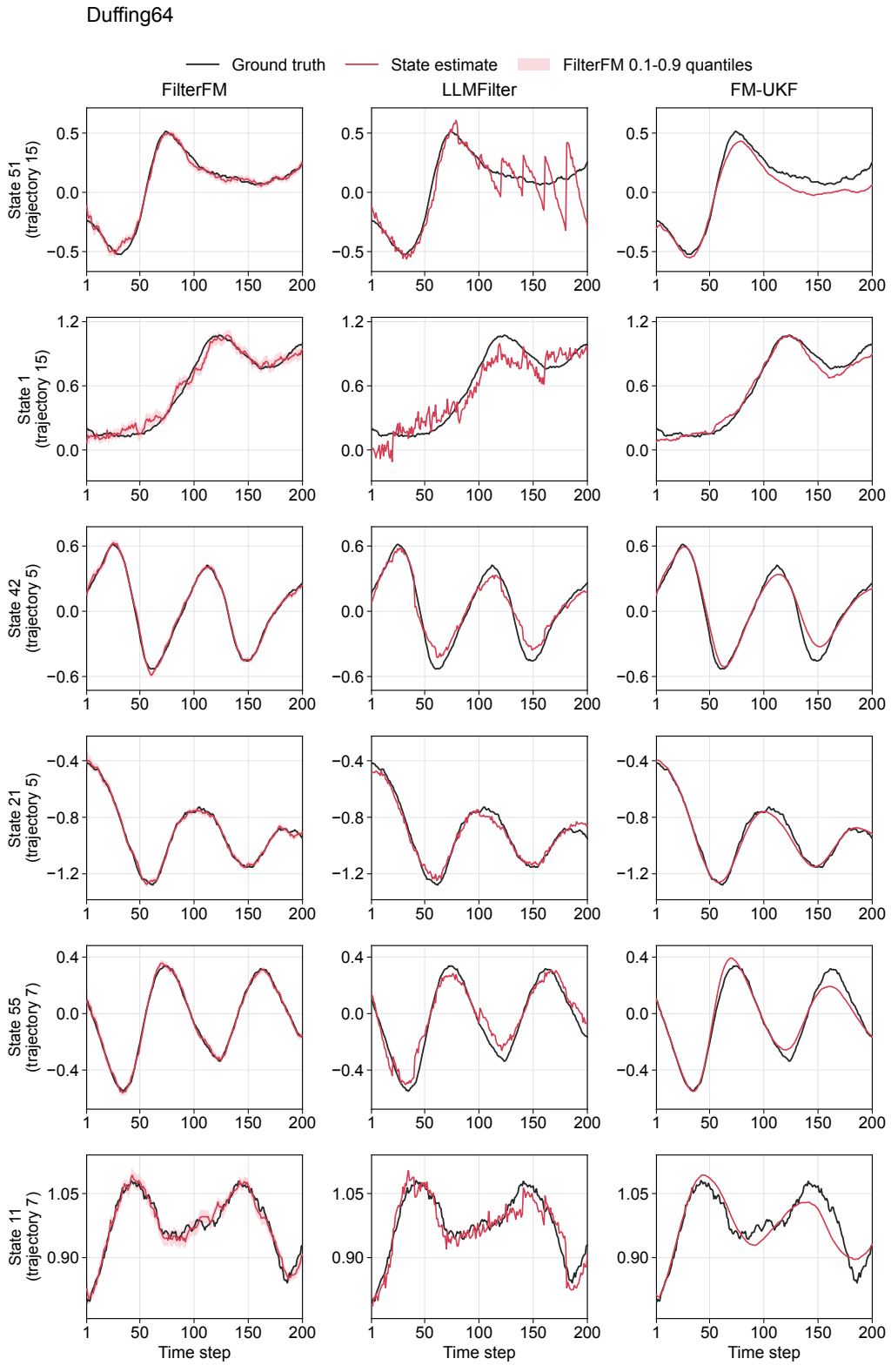

Figure 24: In-domain state estimation on Duffing64: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

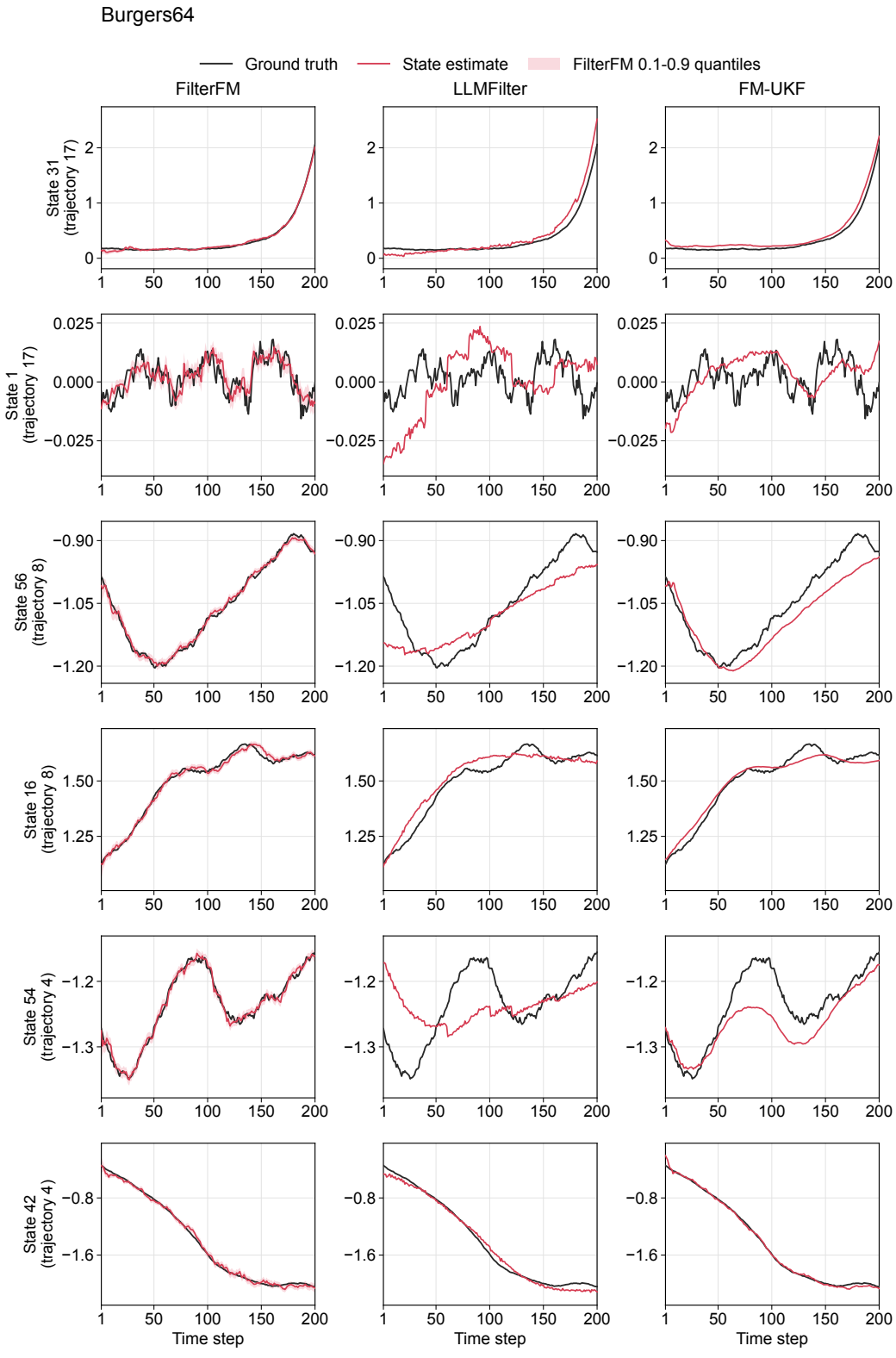

Figure 25: In-domain state estimation on Burgers64: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

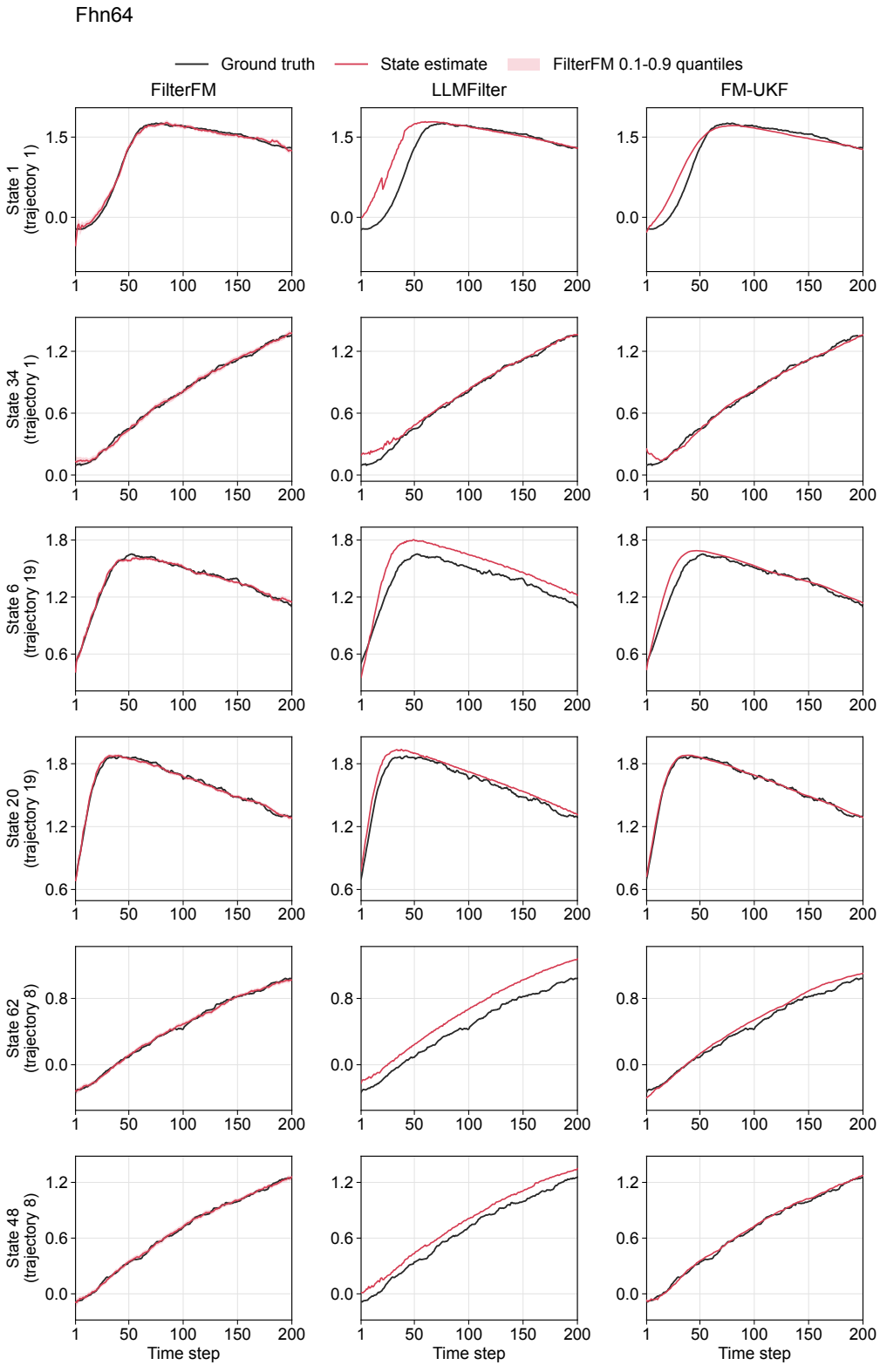

Figure 26: In-domain state estimation on Fhn64: six state channels from three held-out trajectories, with the same axis limits across methods in each row.

E.2 Zero-Shot State Estimation

These Panda systems are generated by skew-product coupling between driver and response dynamics (Lai et al., 2026). The 457 dynamical systems used for zero-shot evaluation are absent from the Panda training sources, including those used for high-dimensional composites. We exhaustively checked all 200-step state windows and verified that no exact duplicates exist between the evaluation set and the low-dimensional Panda or fourteen benchmark datasets.

Figures 27–42 show the main model on 32 such systems: eight randomly selected systems at each dimension from three to six, with one random trajectory per system and no selection by prediction error. For each selected trajectory, we compare the estimated and true states in every channel over 200 time steps. Each figure places two systems side by side and arranges their state channels from top to bottom. Black curves show the true states, red curves show the posterior medians, and light-red bands show the predicted 0.1–0.9 quantile intervals. This view exposes differences in estimation quality across channels, including deviations near abrupt changes and turning points.

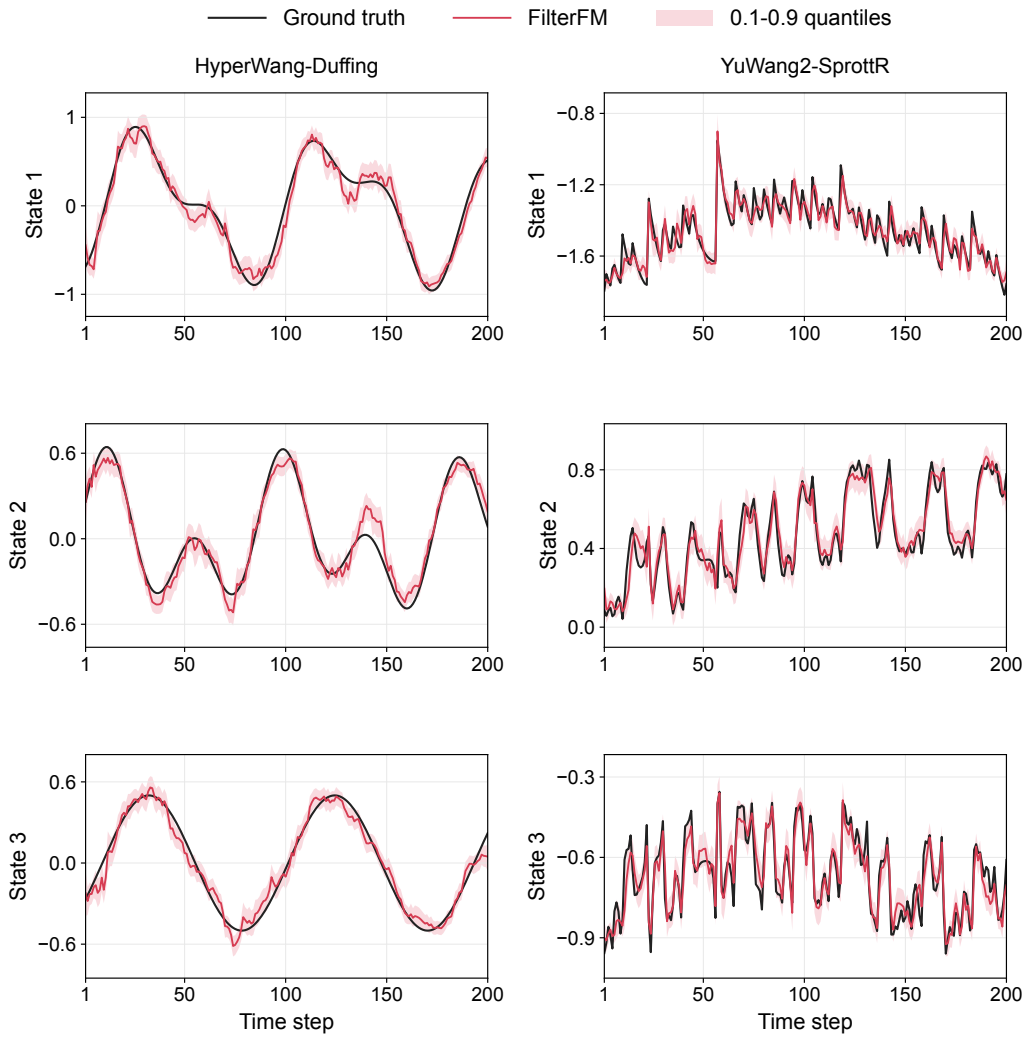

Figure 27: Zero-shot state estimation on two three-dimensional held-out systems. Each column shows one system, with all three state channels arranged from top to bottom. Black and red curves denote the ground truth and FilterFM posterior median, respectively; light-red bands indicate the 0.1–0.9 quantile intervals. All examples use the main checkpoint without fine-tuning.

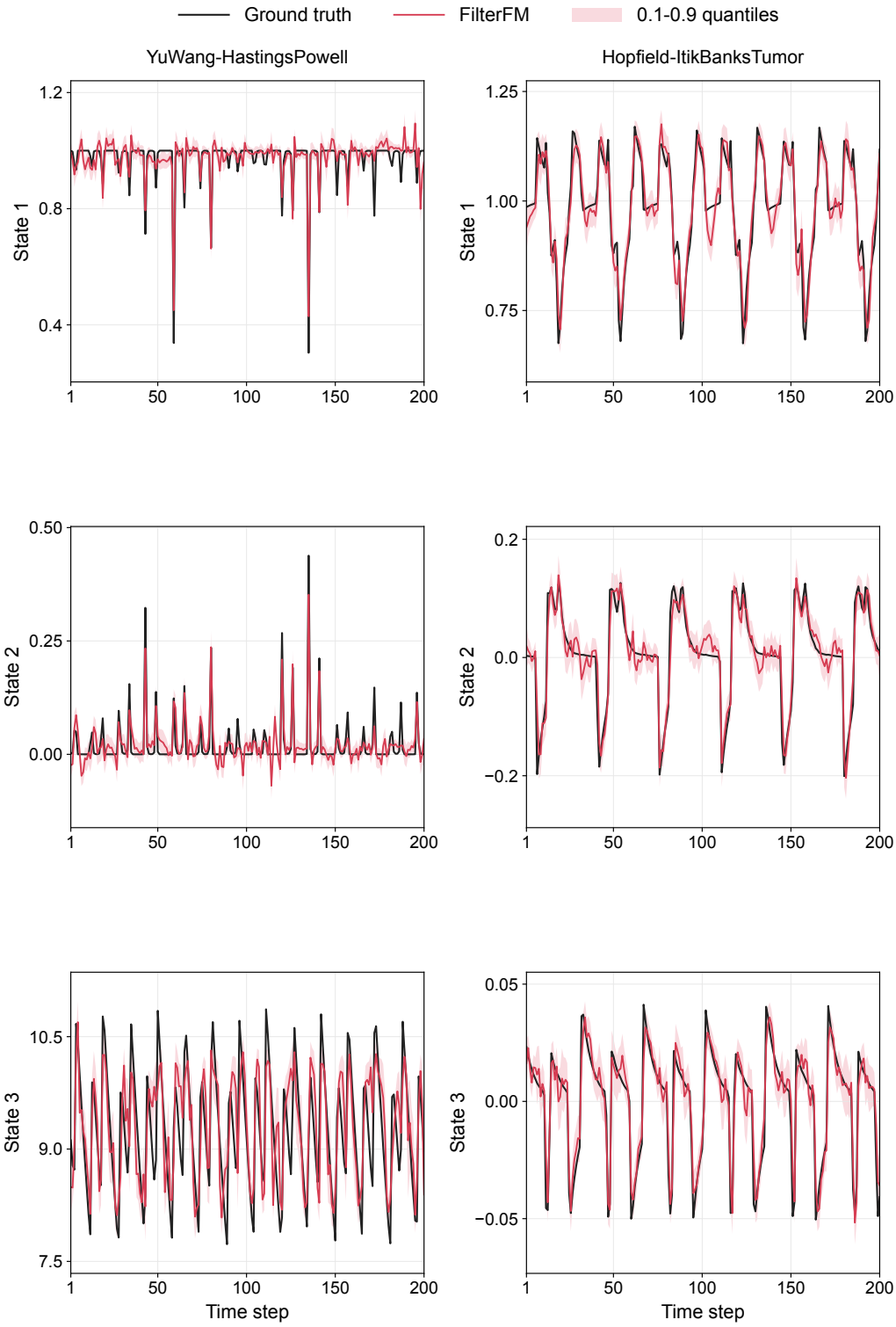

Figure 28: Zero-shot state estimation on two additional three-dimensional held-out systems. Each column shows one system and all three state channels, using the same checkpoint and plotting conventions as Figure 27.

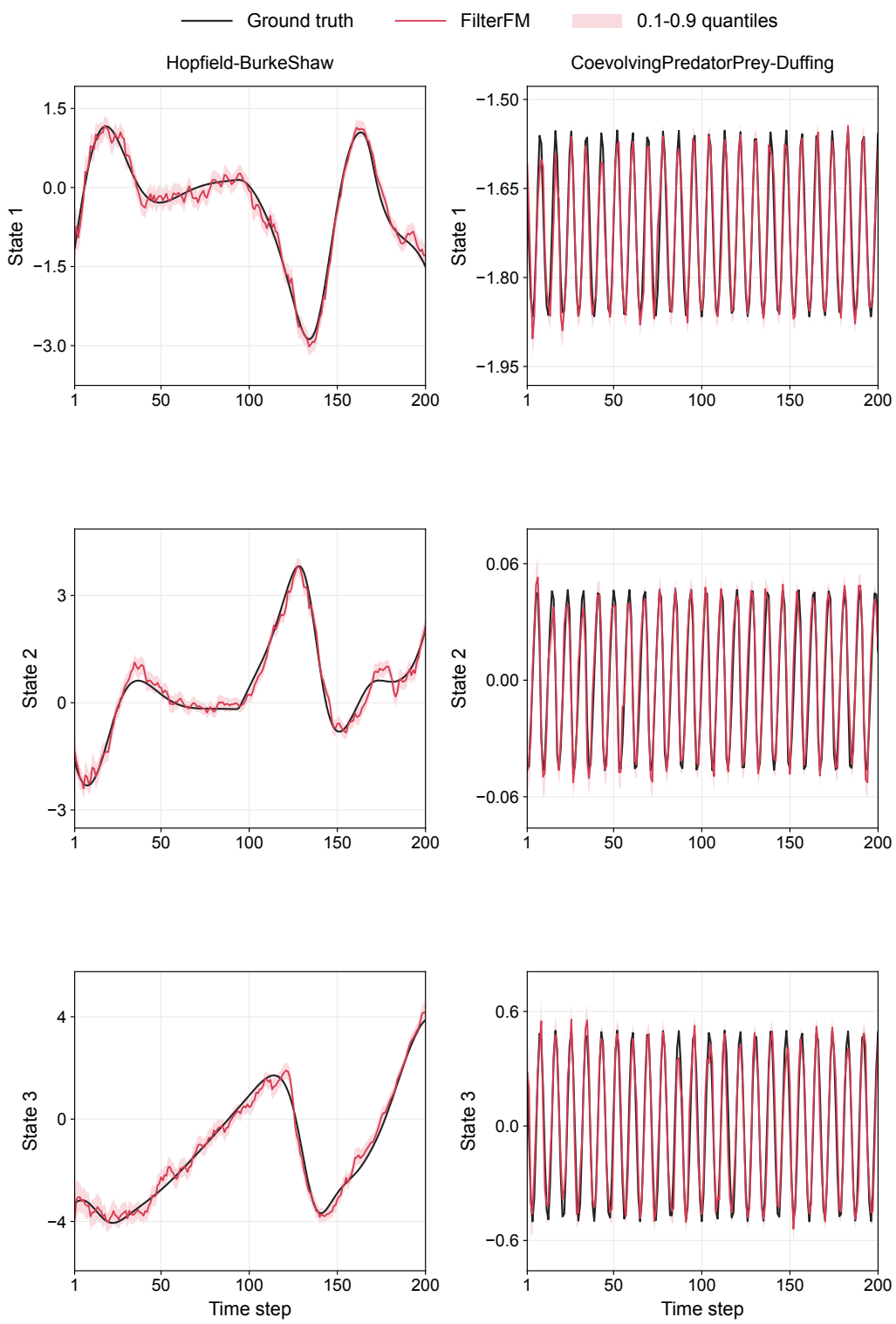

Figure 29: Zero-shot state estimation on two additional three-dimensional held-out systems. Each column shows one system and all three state channels, using the same checkpoint and plotting conventions as Figure 27.

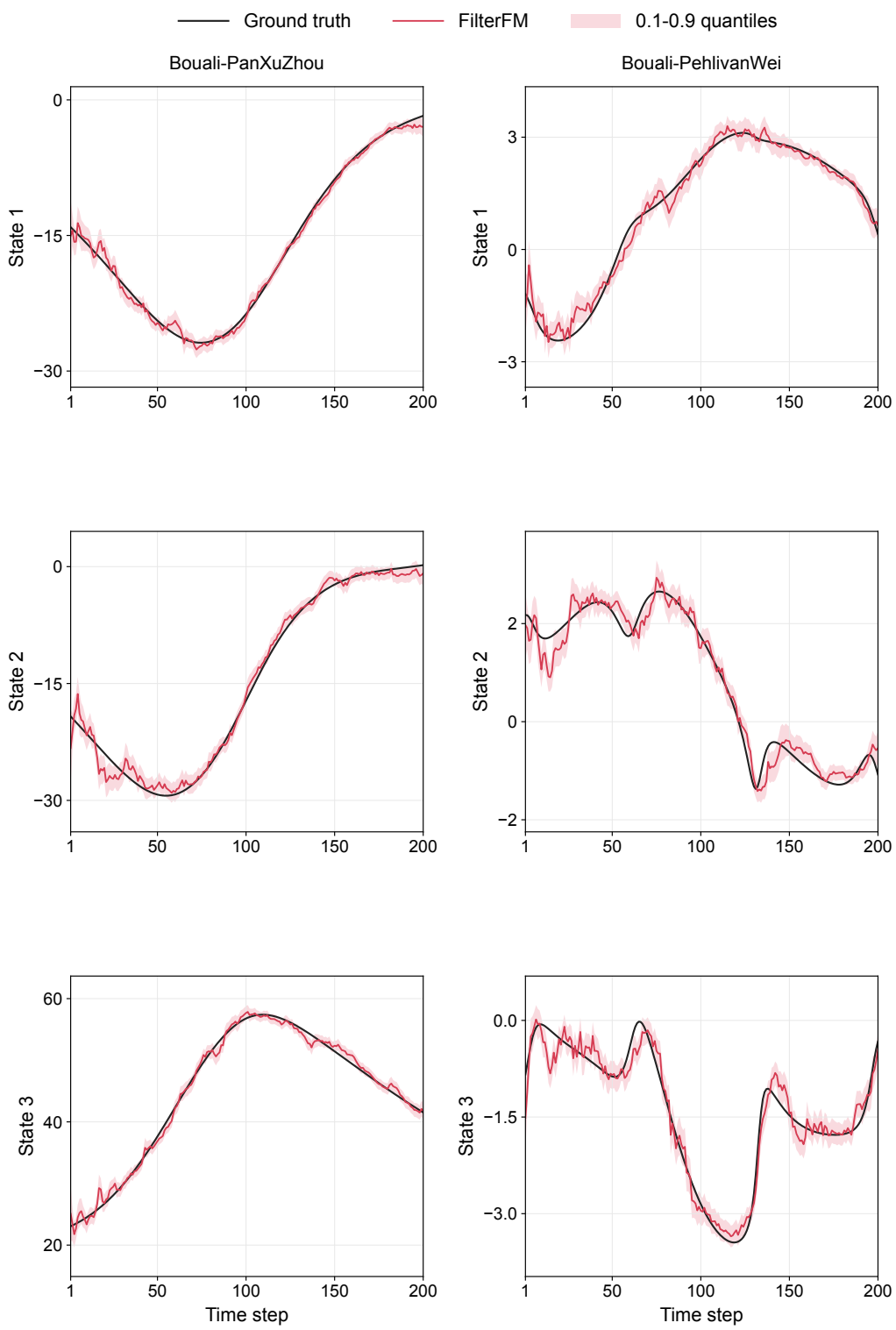

Figure 30: Zero-shot state estimation on two additional three-dimensional held-out systems. Each column shows one system and all three state channels, using the same checkpoint and plotting conventions as Figure 27.

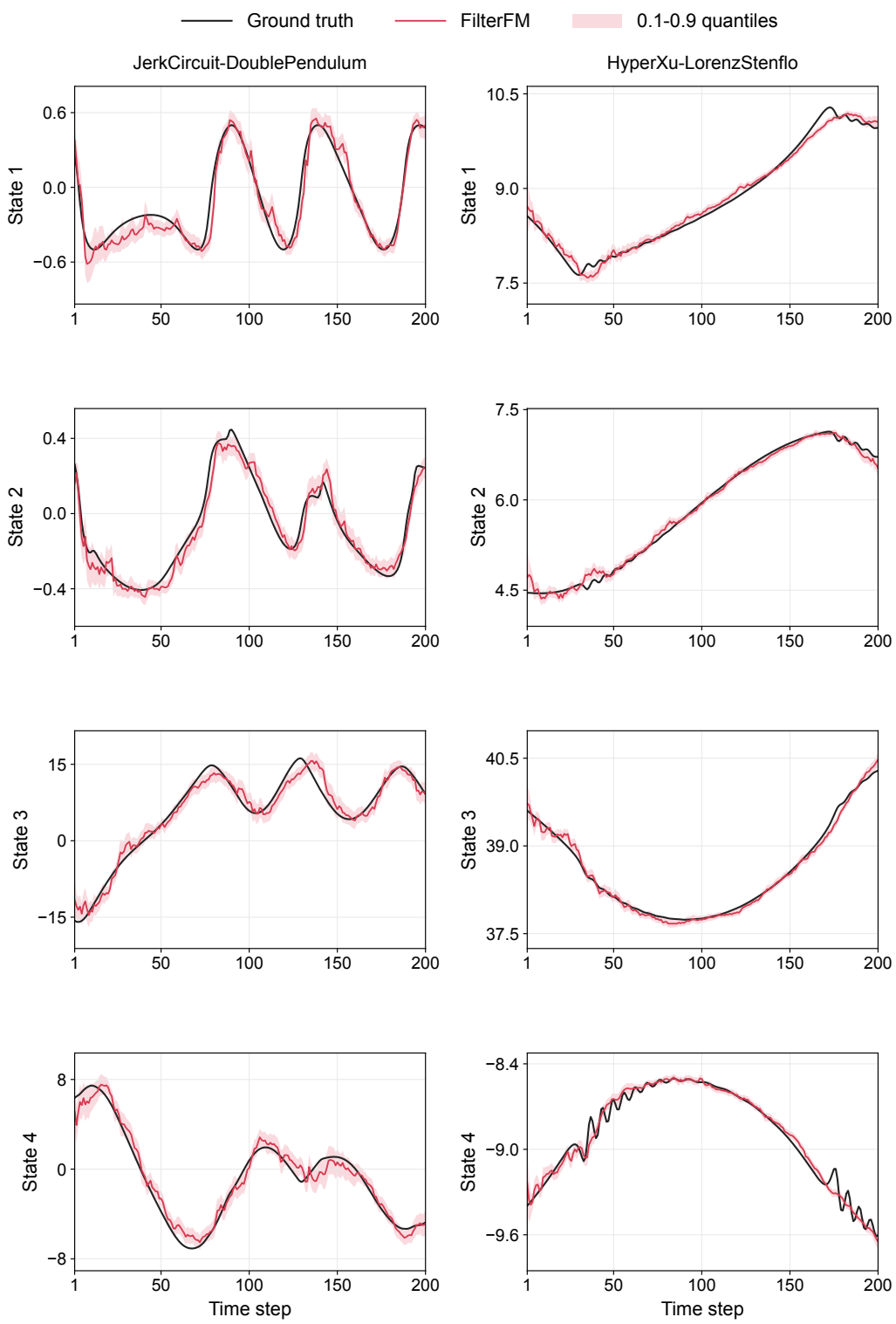

Figure 31: Zero-shot state estimation on two additional four-dimensional held-out systems. Each column shows one system and all four state channels, using the same checkpoint and plotting conventions as Figure 27.

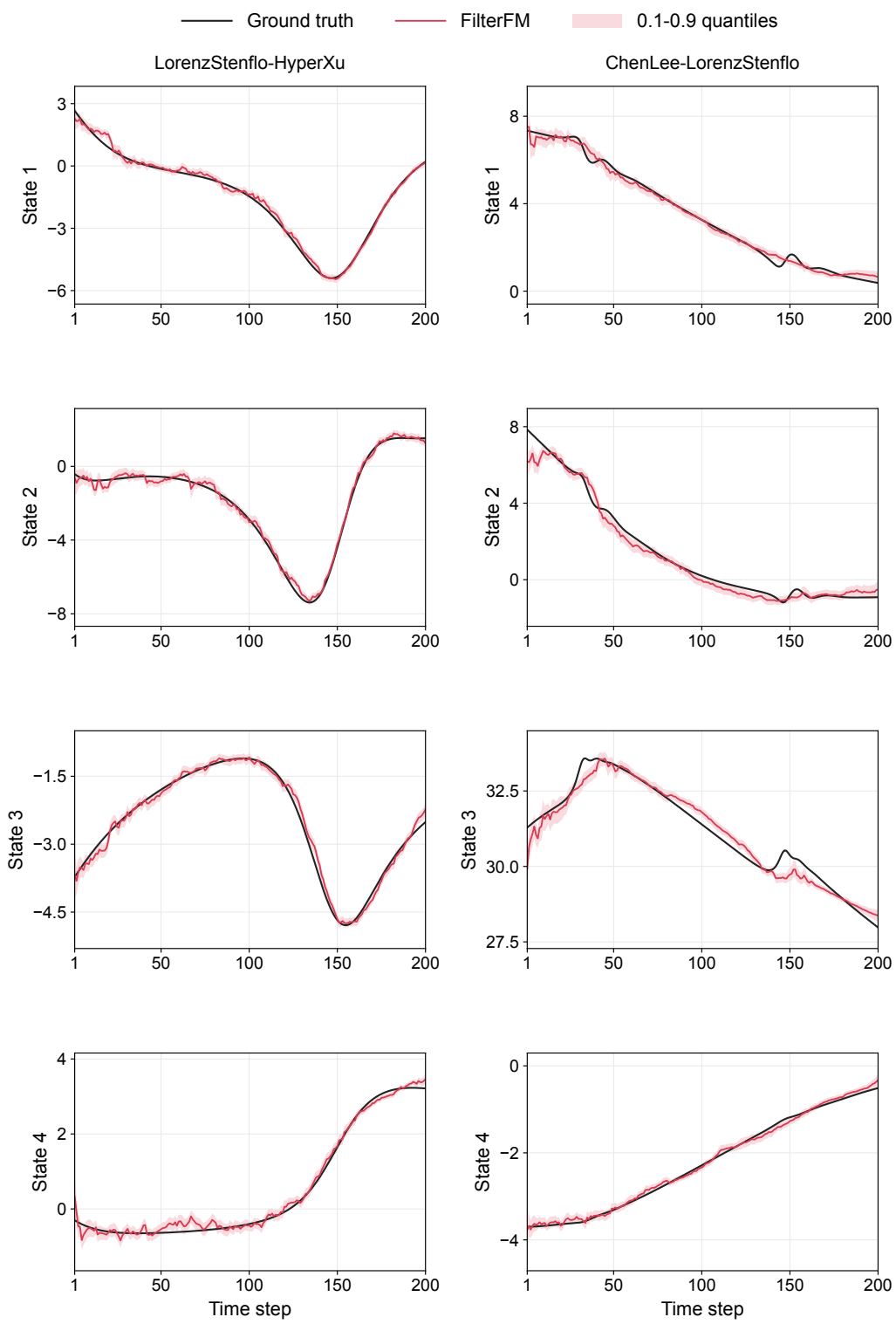

Figure 32: Zero-shot state estimation on two additional four-dimensional held-out systems. Each column shows one system and all four state channels, using the same checkpoint and plotting conventions as Figure 27.

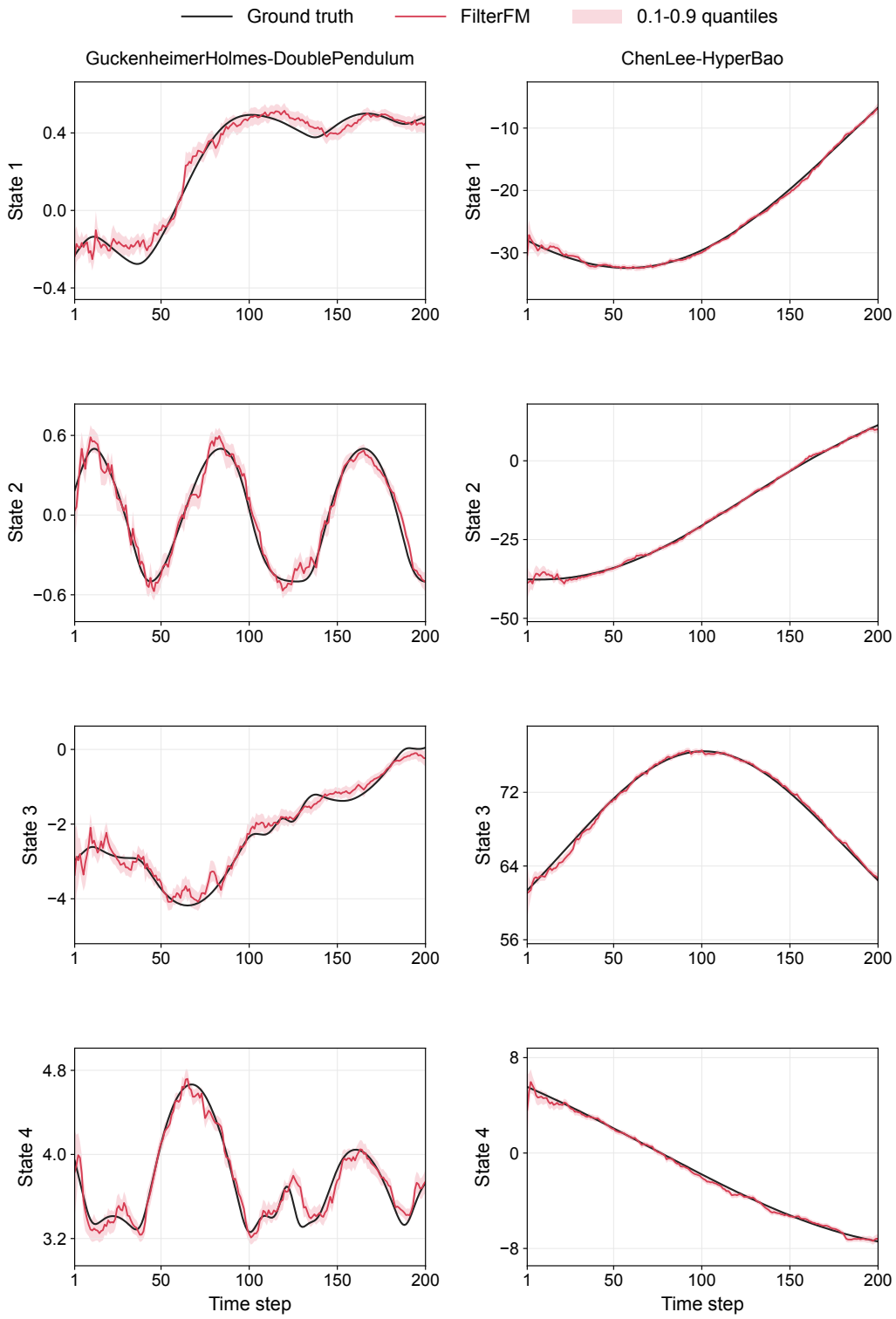

Figure 33: Zero-shot state estimation on two additional four-dimensional held-out systems. Each column shows one system and all four state channels, using the same checkpoint and plotting conventions as Figure 27.

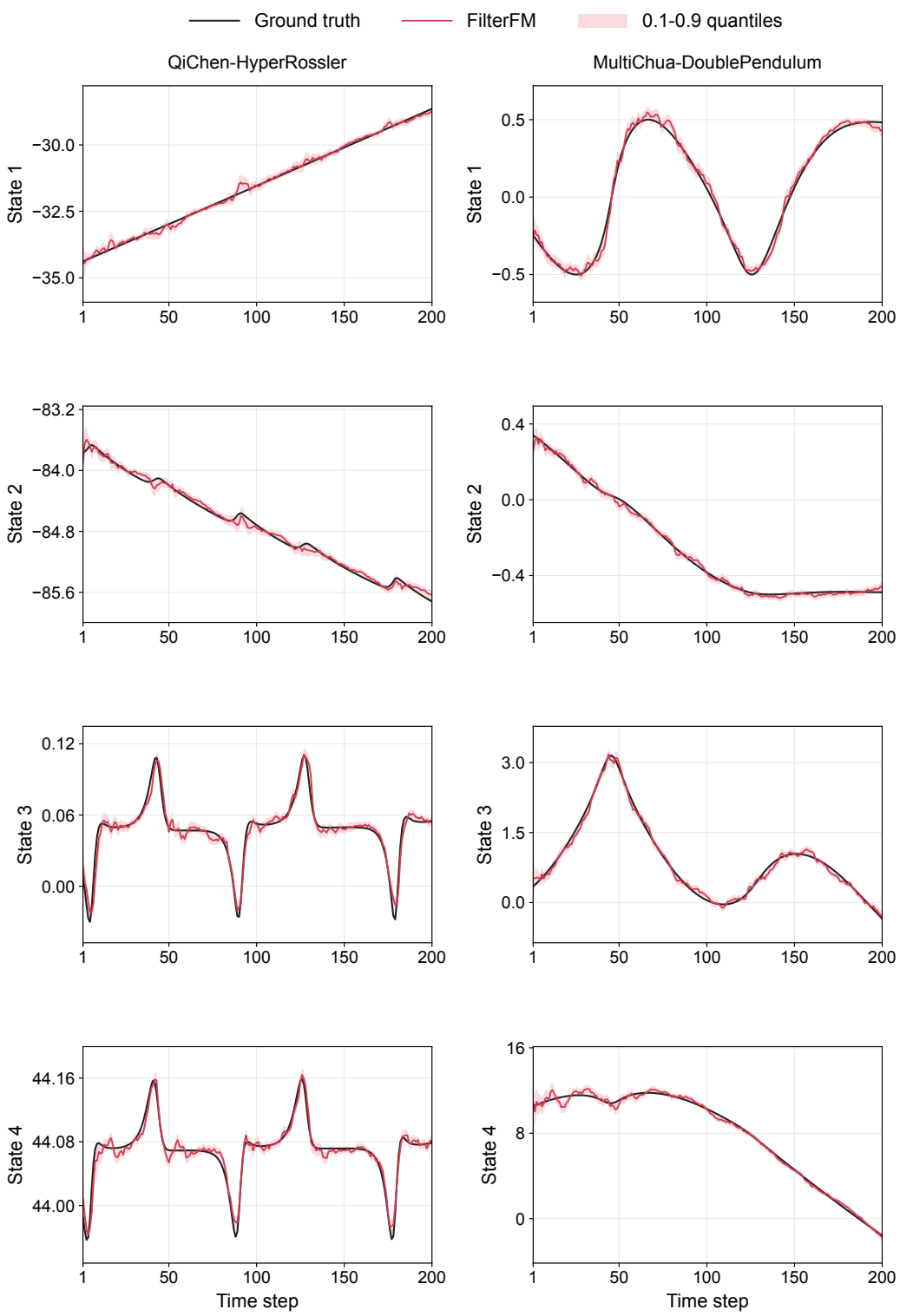

Figure 34: Zero-shot state estimation on two additional four-dimensional held-out systems. Each column shows one system and all four state channels, using the same checkpoint and plotting conventions as Figure 27.

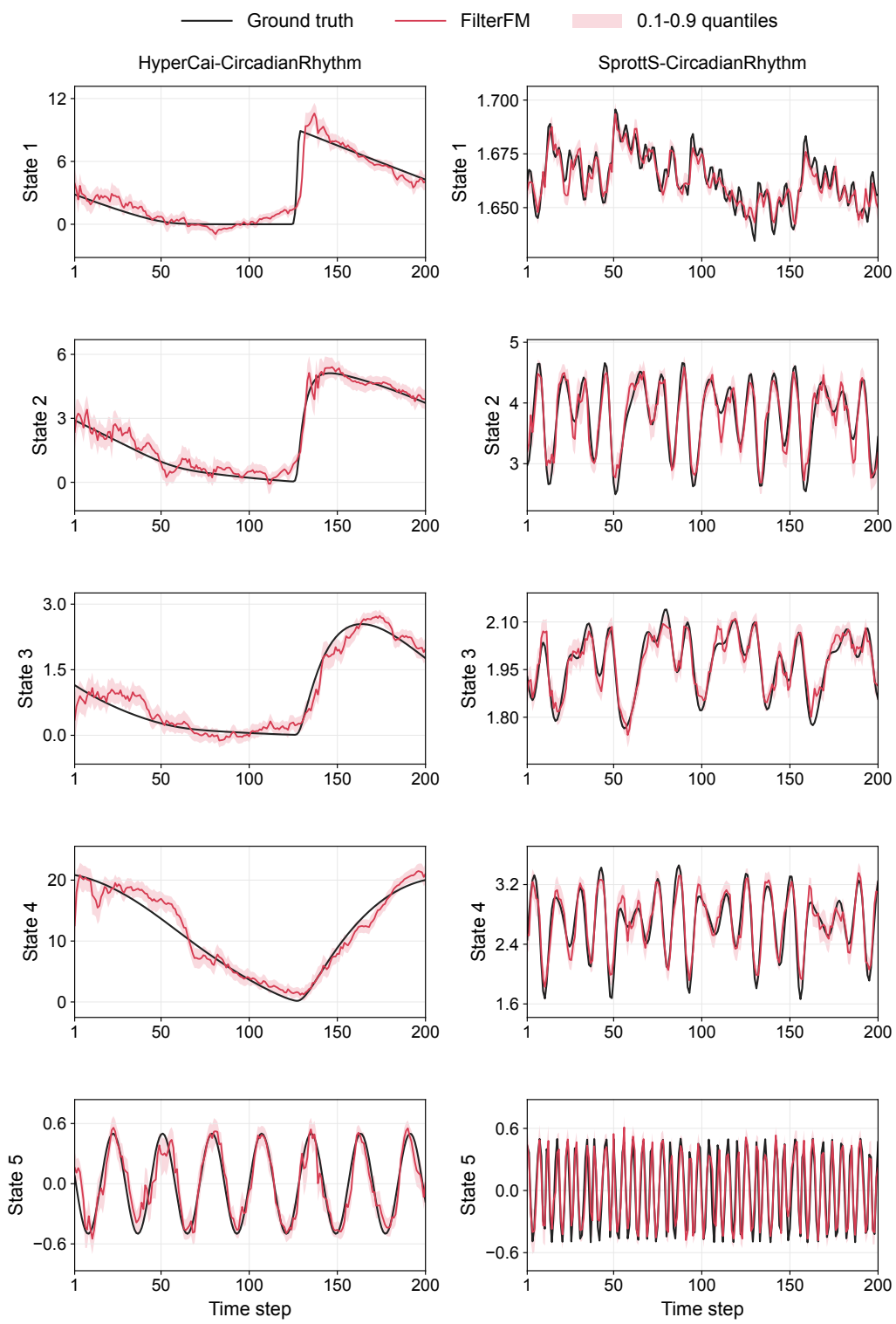

Figure 35: Zero-shot state estimation on two additional five-dimensional held-out systems. Each column shows one system and all five state channels, using the same checkpoint and plotting conventions as Figure 27.

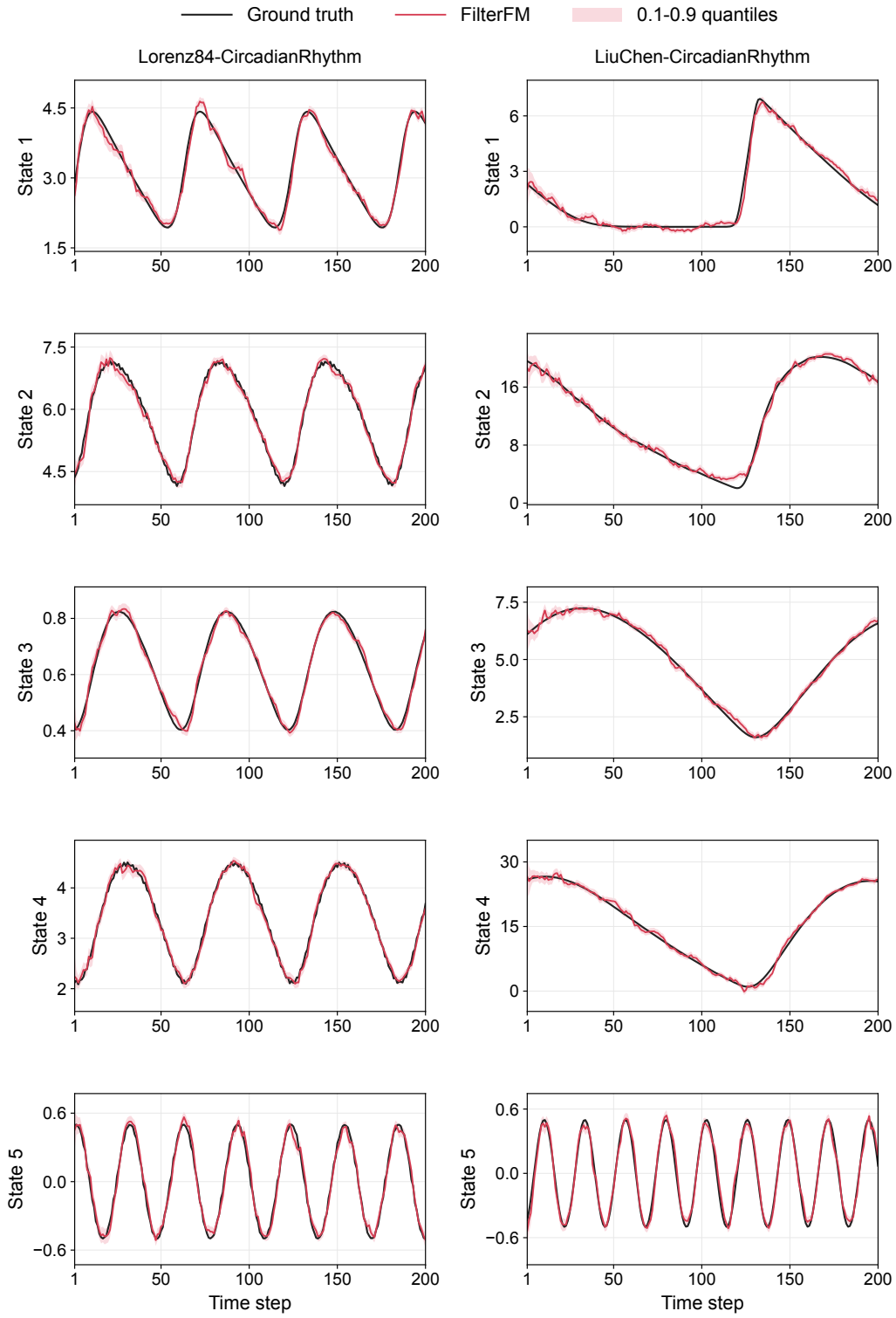

Figure 36: Zero-shot state estimation on two additional five-dimensional held-out systems. Each column shows one system and all five state channels, using the same checkpoint and plotting conventions as Figure 27.

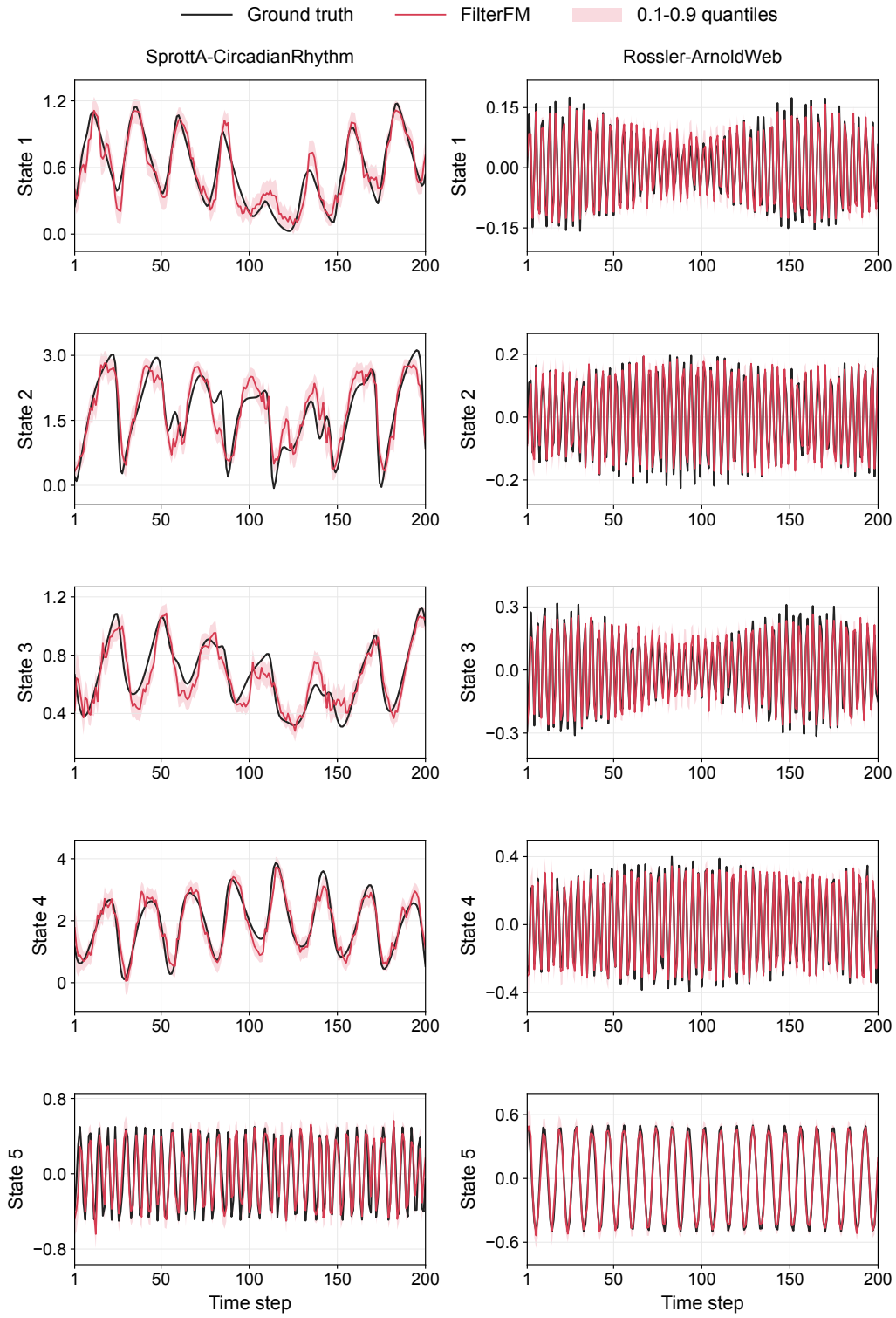

Figure 37: Zero-shot state estimation on two additional five-dimensional held-out systems. Each column shows one system and all five state channels, using the same checkpoint and plotting conventions as Figure 27.

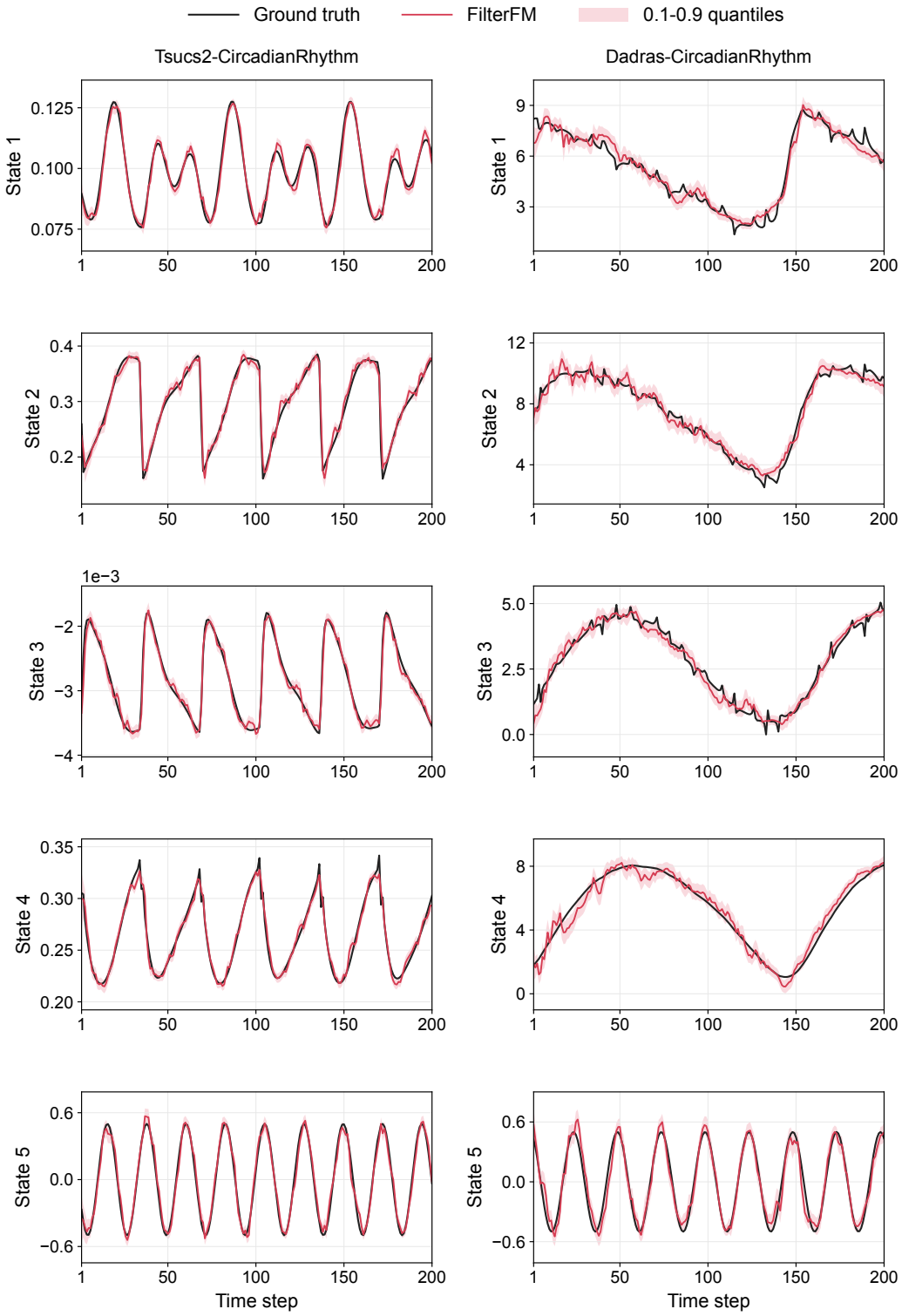

Figure 38: Zero-shot state estimation on two additional five-dimensional held-out systems. Each column shows one system and all five state channels, using the same checkpoint and plotting conventions as Figure 27.

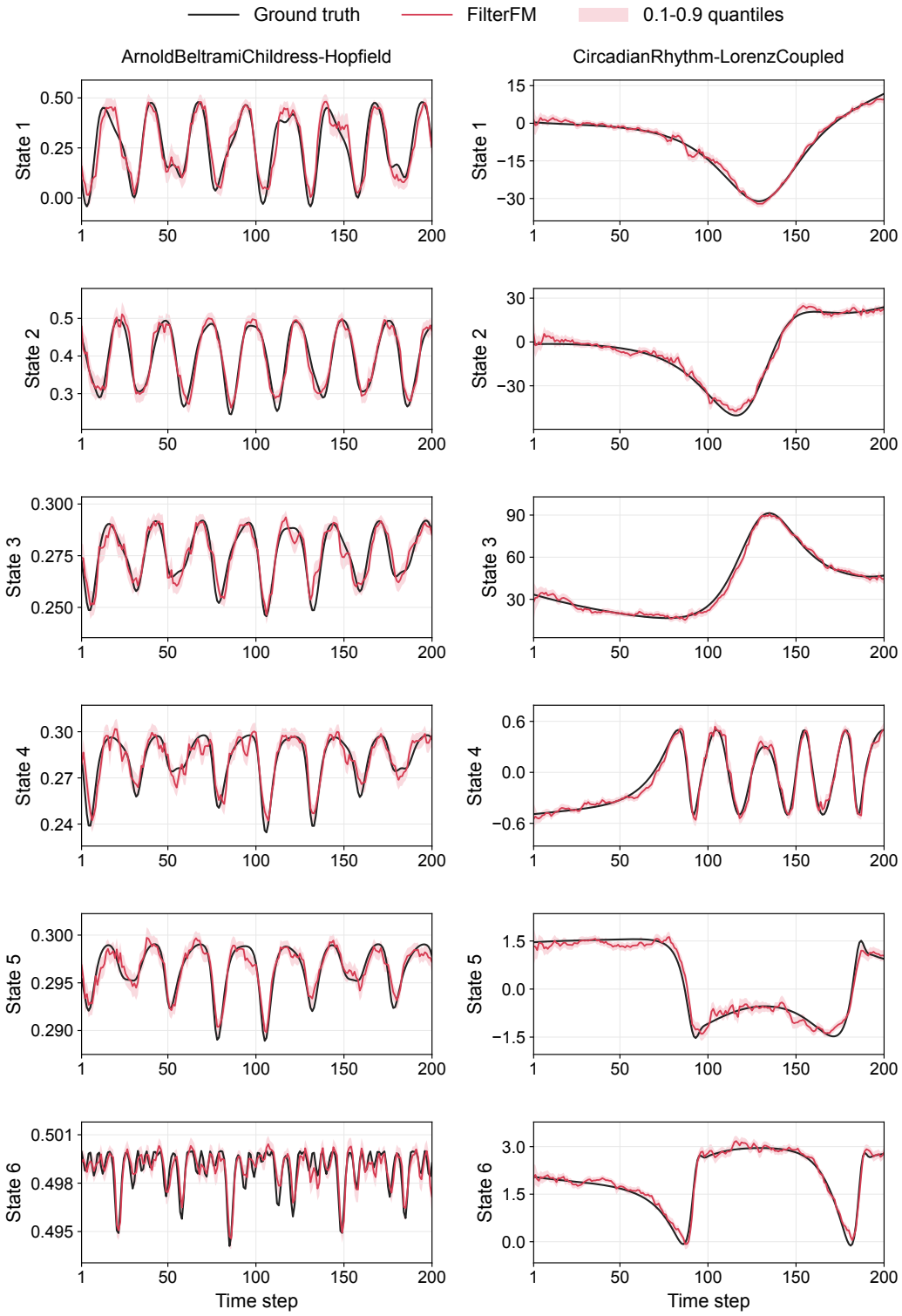

Figure 39: Zero-shot state estimation on two additional six-dimensional held-out systems. Each column shows one system and all six state channels, using the same checkpoint and plotting conventions as Figure 27.

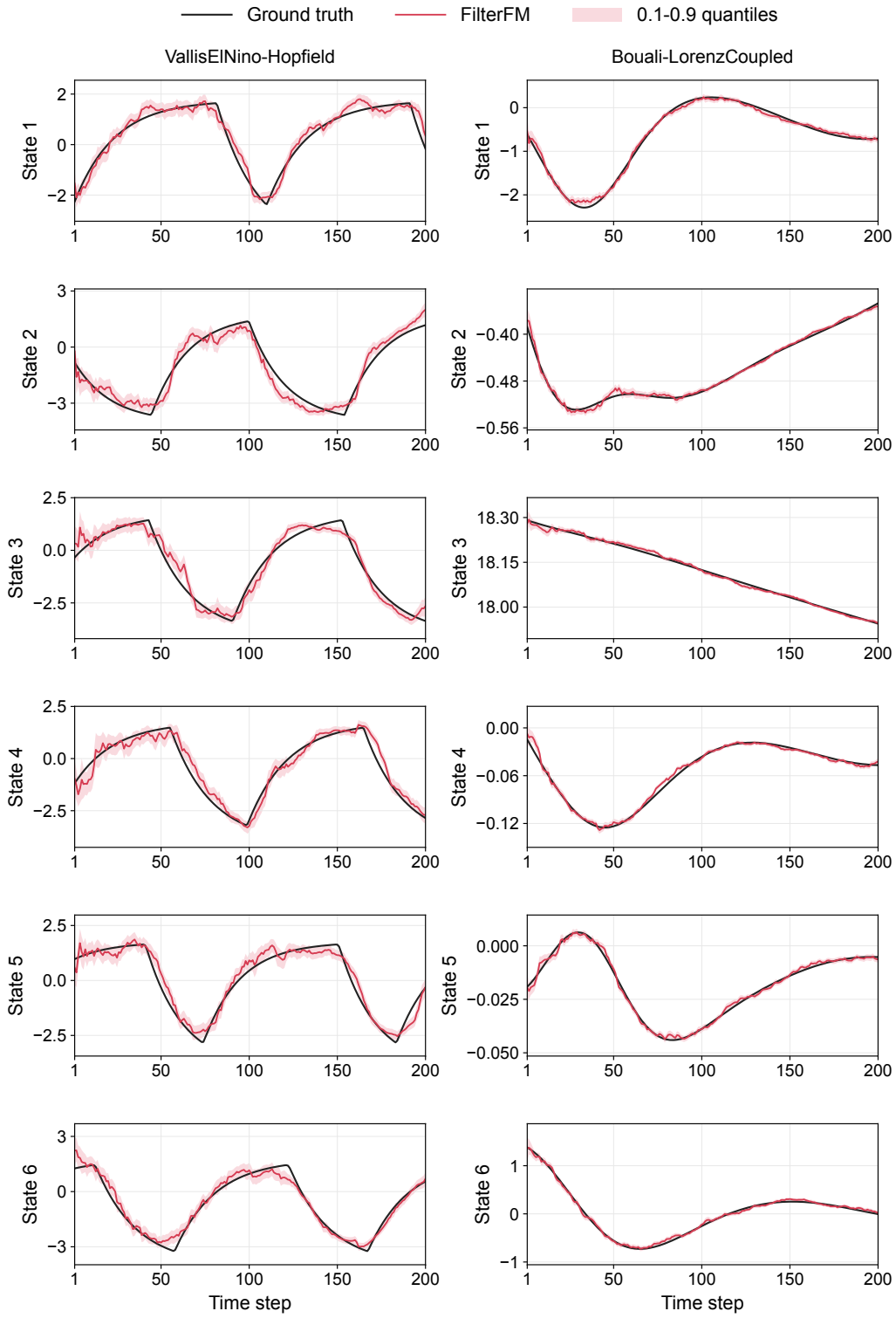

Figure 40: Zero-shot state estimation on two additional six-dimensional held-out systems. Each column shows one system and all six state channels, using the same checkpoint and plotting conventions as Figure 27.

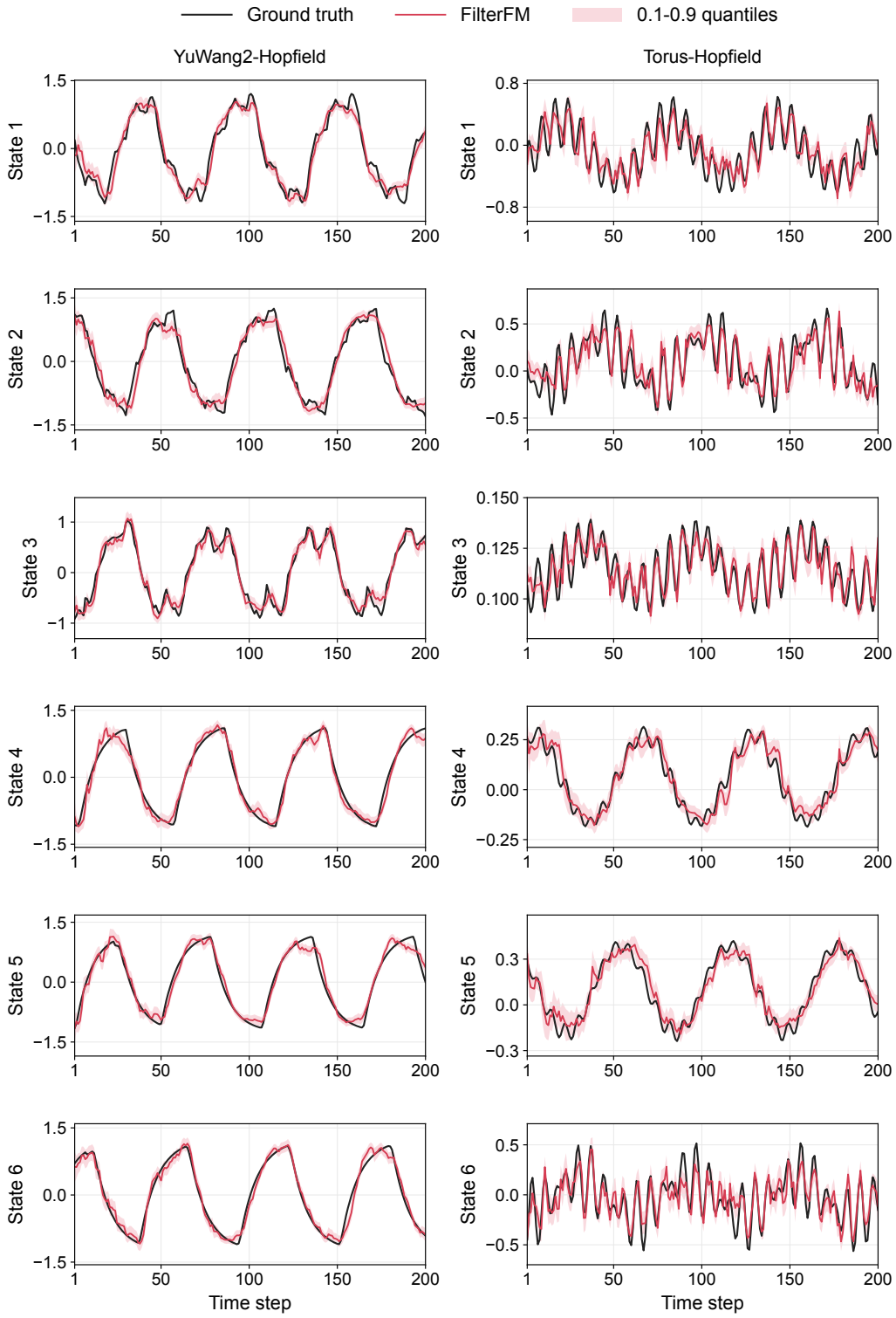

Figure 41: Zero-shot state estimation on two additional six-dimensional held-out systems. Each column shows one system and all six state channels, using the same checkpoint and plotting conventions as Figure 27.

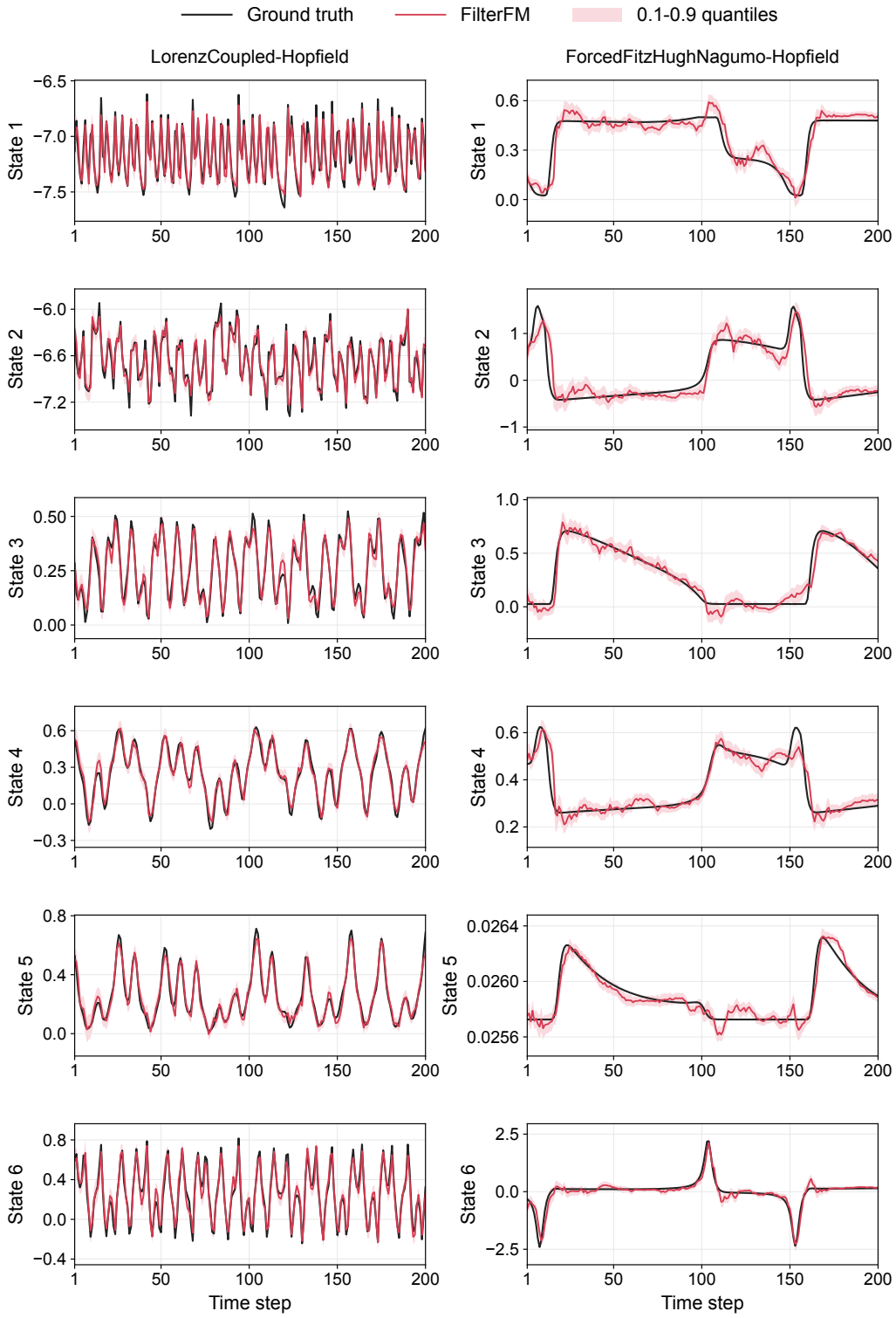

Figure 42: Zero-shot state estimation on two additional six-dimensional held-out systems. Each column shows one system and all six state channels, using the same checkpoint and plotting conventions as Figure 27.

F System-as-Prompt Inputs for LLMFilter

The LLMFilter code repository does not include its task-specific System-as-Prompt texts, so we construct a prompt for each of our fourteen benchmark systems following the format described by Liu et al. (2025). Each prompt first specifies the filtering task and introduces the underlying dynamical system and observation model, then describes the dataset, including its trajectory length, number of trajectories, and observation-noise variance. A short excerpt of paired observation and state trajectories illustrates the input and target sequences, followed by a final instruction to recover the hidden states from noisy observations. We provide the resulting prompts below.

Tracking

```
[System-as-Prompt]
Dataset name: tracking
System: tracking

[Task]
Estimate the 4-dimensional hidden state trajectory from the 4-dimensional noisy observation
     trajectory.
This is a linear target-tracking system with two-dimensional position and velocity.
x=[p_x,p_y,v_x,v_y]. The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 10. Observation noise standard deviation: 3.16228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 4 dimensions:
[[-2.3365, 3.651, -0.6574, -0.199], [2.4867, -0.1238, -2.3462, 0.1804], [-1.5614, -3.5142, 6.1511,
     -2.7935], [-1.2404, 2.3006, 6.8164, 3.4445], [-2.4679, -0.0331, -0.5486, -5.8093]]
States x[0:5] first 4 dimensions:
[[0.2666, 0.1125, 0.2957, 0.8013], [-0.0597, -0.2589, 0.4677, 0.7646], [0.473, -0.9342, 0.8665,
     0.9388], [0.0318, -1.2292, 0.6055, 0.8531], [0.5262, -1.0128, 0.6046, 1.0829]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Double-Pendulum

```
[System-as-Prompt]
Dataset name: pendulum
System: pendulum

[Task]
Estimate the 4-dimensional hidden state trajectory from the 4-dimensional noisy observation
     trajectory.
This is a full-observation double-pendulum filtering system.
x=[theta1,omega1,theta2,omega2]. The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.1. Observation noise standard deviation: 0.316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 4 dimensions:
[[2.3351, 0.0744, 0.5731, -0.2456], [2.2368, -0.342, -0.4001, 0.1449], [2.6062, 0.3229, 0.2416,
     -0.2461], [1.6795, -0.368, 0.5794, -0.2177], [1.5603, -0.0839, -0.0372, -0.124]]
States x[0:5] first 4 dimensions:
[[2.0957, 0.0491, 0.056, 0.0093], [2.0841, -0.0597, 0.0549, -0.0477], [2.0788, -0.153, 0.0652,
     -0.103], [2.0652, -0.2563, 0.0682, -0.1469], [2.0631, -0.3582, 0.0552, -0.1739]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Selkov

```
[System-as-Prompt]
Dataset name: selkov
```

```
System: selkov

[Task]
Estimate the 2-dimensional hidden state trajectory from the 2-dimensional noisy observation
     trajectory.
This is the selkov dynamical system used in the filtering benchmark.
The observation is generated by adding Gaussian noise to the system state trajectory.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.001. Observation noise standard deviation: 0.0316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 2 dimensions:
[[0.053, -0.0187], [0.04, 0.0077], [0.042, 0.0424], [0.0299, 0.0528], [-0.0019, 0.1154]]
States x[0:5] first 2 dimensions:
[[0.0147, 0.0338], [0.0216, 0.0333], [0.0218, 0.0395], [0.0199, 0.0552], [0.0248, 0.0692]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Oscillator

```
[System-as-Prompt]
Dataset name: oscillator
System: oscillator

[Task]
Estimate the 2-dimensional hidden state trajectory from the 2-dimensional noisy observation
     trajectory.
This is the oscillator dynamical system used in the filtering benchmark.
The observation is generated by adding Gaussian noise to the system state trajectory.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.001. Observation noise standard deviation: 0.0316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 2 dimensions:
[[-0.076, 0.1565], [-0.092, 0.1361], [-0.0751, 0.0438], [-0.1202, 0.1017], [-0.078, 0.033]]
States x[0:5] first 2 dimensions:
[[-0.0731, 0.1101], [-0.079, 0.1123], [-0.0931, 0.0897], [-0.0843, 0.0815], [-0.1151, 0.0763]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Hopf

```
[System-as-Prompt]
Dataset name: hopf
System: hopf

[Task]
Estimate the 2-dimensional hidden state trajectory from the 2-dimensional noisy observation
     trajectory.
This is the hopf dynamical system used in the filtering benchmark.
The observation is generated by adding Gaussian noise to the system state trajectory.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.001. Observation noise standard deviation: 0.0316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 2 dimensions:
[[-0.134, -0.0344], [-0.0857, -0.0142], [-0.1163, -0.0468], [-0.1859, 0.0139], [-0.0769, -0.0572]]
States x[0:5] first 2 dimensions:
[[-0.1139, -0.0391], [-0.112, -0.0331], [-0.1361, -0.0311], [-0.1238, -0.0145], [-0.1108, -0.0198]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Lorenz96

```
[System-as-Prompt]
Dataset name: lorenz96
System: lorenz96

[Task]
Estimate the 72-dimensional hidden state trajectory from the 72-dimensional noisy observation
     trajectory.
This is the lorenz96 dynamical system used in the filtering benchmark.
The observation is generated by adding Gaussian noise to the system state trajectory.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 10. Observation noise standard deviation: 3.16228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[7.217, 11.8028, 5.9928, 10.1124, 3.2039, 11.649, 10.5332, 10.3298], [8.6512, 5.9891, 13.2407,
     11.0471, 6.6574, 4.7631, 14.4466, 9.552], [6.6121, 7.2576, 13.0835, 3.2558, 12.8608, 5.6326,
     13.7519, 6.9048], [4.835, 9.5169, 7.4263, 6.1472, 13.886, 6.9219, 6.0175, 7.2279], [5.8587,
     5.9673, 11.4877, 9.92, 9.5799, 6.3204, 6.1359, 0.8301]]
States x[0:5] first 8 dimensions:
[[8.2484, 8.6009, 9.4624, 7.769, 9.0011, 7.563, 11.1538, 7.2436], [8.1843, 8.7587, 9.4068, 7.8009,
     8.8663, 7.8609, 10.9826, 7.3506], [8.1167, 8.8817, 9.3587, 7.7977, 8.7446, 8.1356, 10.8225,
     7.3758], [8.0818, 9.0083, 9.3176, 7.7934, 8.6628, 8.406, 10.6918, 7.3546], [8.0481, 9.1408,
     9.2835, 7.7486, 8.6165, 8.6333, 10.553, 7.3134]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Vissio–Lucarini 20 (VL20)

```
[System-as-Prompt]
Dataset name: VL20
System: VL20

[Task]
Estimate the 72-dimensional hidden state trajectory from the 72-dimensional noisy observation
     trajectory.
This is the VL20 dynamical system used in the filtering benchmark.
The observation is generated by adding Gaussian noise to the system state trajectory.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 1. Observation noise standard deviation: 1.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[10.4613, 10.6931, 11.8252, 10.1098, 13.687, 10.0067, 10.3651, 8.97], [9.5676, 11.5139, 9.2724,
     11.1337, 10.9698, 11.0236, 9.9012, 9.6793], [9.9129, 10.0087, 12.4593, 9.302, 12.2774,
     10.4409, 10.1498, 9.4668], [12.5091, 10.7382, 8.6484, 12.31, 9.779, 10.8129, 9.5976, 9.7819],
     [11.6314, 10.1489, 10.6527, 12.0558, 11.8657, 9.5705, 8.3921, 11.0744]]
States x[0:5] first 8 dimensions:
[[11.1177, 10.4969, 11.1051, 10.6981, 12.2874, 10.7532, 10.2626, 10.4685], [11.1292, 10.5212,
     10.9451, 10.7641, 12.1162, 10.5585, 9.9415, 10.259], [11.1767, 10.5243, 10.7736, 10.8244,
     11.9529, 10.321, 9.6314, 10.0568], [11.2339, 10.5174, 10.6125, 10.8467, 11.7753, 10.0564,
     9.337, 9.8794], [11.2931, 10.4821, 10.4513, 10.8361, 11.5837, 9.7627, 9.0667, 9.7384]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

LorenzUV40

```
[System-as-Prompt]
Dataset name: lorenzUV40
System: lorenzUV40

[Task]
Estimate the 40-dimensional hidden state trajectory from the 40-dimensional noisy observation
     trajectory.
This is a 40-dimensional multiscale Lorenz UV chaotic field.
```

```
The observation is the full 40-dimensional state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.1. Observation noise standard deviation: 0.316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[1.1838, -0.2644, -0.4046, 0.392, -0.0424, 0.3177, 0.5152, -0.2839], [1.1976, -0.2957, 0.0965,
     -0.3751, -0.3243, 0.2997, 0.3714, 0.0995], [1.4008, 0.0777, -0.3103, 0.216, 0.2011, 0.2045,
     -0.176, 0.4571], [0.6348, -0.0176, 0.6579, 0.4182, 0.6598, 0.4496, 0.474, 0.7679], [1.2491,
     0.4174, 0.5635, 0.4596, 0.4818, 0.549, -0.1463, 0.3737]]
States x[0:5] first 8 dimensions:
[[1.0069, -0.0089, 0.0022, -0.0152, -0.0065, -0.0062, 0.0107, 0.0227], [1.1028, 0.0903, 0.1019,
     0.0848, 0.0945, 0.0935, 0.11, 0.122], [1.1962, 0.1903, 0.2009, 0.1844, 0.1935, 0.1928, 0.2088,
      0.2214], [1.2891, 0.2904, 0.299, 0.2837, 0.2924, 0.2919, 0.3055, 0.3195], [1.3821, 0.3909,
     0.3967, 0.3817, 0.3906, 0.3897, 0.4014, 0.4175]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Vanderpol64

```
[System-as-Prompt]
Dataset name: vanderpol_chain64
System: vanderpol_chain

[Task]
Estimate the 64-dimensional hidden state trajectory from the 64-dimensional noisy observation
     trajectory.
This is a 64-dimensional coupled Van der Pol oscillator chain.
The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.1. Observation noise standard deviation: 0.316228.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[-0.1152, -0.4354, 0.3595, -0.0196, 0.827, 0.0096, 0.3698, 0.7364], [0.0039, 0.3205, 0.4337,
     0.4082, 0.5784, -0.0598, 0.8918, 0.6051], [-0.0024, 0.4503, -0.2808, 0.7159, 0.3619, 0.1796,
     0.2246, 0.7126], [0.3753, -0.3373, 0.3981, 0.9796, 0.8325, 0.3278, 0.1004, 0.2375], [-0.3567,
     0.012, -0.056, -0.239, 0.1678, 0.4754, 0.6086, 0.201]]
States x[0:5] first 8 dimensions:
[[-0.1518, 0.137, 0.2615, 0.4188, 0.4666, 0.399, 0.4625, 0.6721], [-0.1422, 0.1333, 0.2596, 0.4127,
      0.4684, 0.3993, 0.4627, 0.6723], [-0.1344, 0.1312, 0.2545, 0.4095, 0.4659, 0.4026, 0.4555,
     0.6625], [-0.1273, 0.1163, 0.245, 0.4086, 0.4653, 0.4008, 0.4563, 0.6612], [-0.1196, 0.1122,
     0.2385, 0.4013, 0.4607, 0.4004, 0.457, 0.6553]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Rossler96

```
[System-as-Prompt]
Dataset name: rossler_chain96
System: rossler_chain

[Task]
Estimate the 96-dimensional hidden state trajectory from the 96-dimensional noisy observation
     trajectory.
This is a 96-dimensional coupled Rossler oscillator chain.
The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 1. Observation noise standard deviation: 1.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
```

```
[[2.6266, 0.2603, 0.6545, 0.6688, 0.5958, -0.24, 0.73, -0.3569], [2.2572, 0.4585, 0.5631, 0.848,
     0.5637, -0.1731, 0.668, -0.1106], [2.1732, 0.1592, 0.5943, 0.8182, 0.7827, -0.1637, 0.9692,
     -0.0537], [2.127, 0.1528, 0.5856, 0.7549, 0.6971, -0.214, 0.7874, 0.0497], [2.1342, 0.4623,
     0.5173, 0.5503, 0.3153, -0.1999, 0.5255, -0.0621]]
States x[0:5] first 8 dimensions:
[[2.27, 0.1798, 0.6545, 0.6492, 0.5901, -0.2364, 0.7453, -0.1839], [2.2677, 0.2485, 0.6257, 0.6499,
      0.601, -0.1437, 0.7632, -0.1358], [2.2289, 0.3511, 0.5929, 0.6424, 0.5865, -0.1602, 0.7227,
     -0.1384], [2.1879, 0.4147, 0.542, 0.6538, 0.5265, -0.1768, 0.7641, -0.0948], [2.1217, 0.4675,
     0.5206, 0.6716, 0.494, -0.1929, 0.7124, -0.0998]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Duffing64

```
[System-as-Prompt]
Dataset name: duffing_chain64
System: duffing_chain

[Task]
Estimate the 64-dimensional hidden state trajectory from the 64-dimensional noisy observation
     trajectory.
This is a 64-dimensional coupled Duffing oscillator chain.
The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.08. Observation noise standard deviation: 0.282843.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[0.0318, 0.1016, 0.2634, 0.7743, 0.5839, 0.5173, 0.7688, 0.463], [-0.1567, 0.2711, 0.1155, 0.7753,
      0.587, 0.4291, 0.7458, 0.4026], [-0.12, 0.3527, 0.0243, 0.8229, 0.5162, 0.5074, 0.7431,
     0.3739], [-0.1524, 0.1966, 0.0326, 0.8224, 0.5549, 0.5074, 0.7418, 0.4085], [0.1344, 0.2404,
     0.0289, 0.8231, 0.5381, 0.4956, 0.7097, 0.3918]]
States x[0:5] first 8 dimensions:
[[-0.0519, 0.204, 0.0703, 0.7993, 0.5544, 0.4901, 0.747, 0.4584], [-0.0742, 0.2113, 0.0484, 0.8018,
      0.5482, 0.4905, 0.7386, 0.44], [-0.0834, 0.2206, 0.0386, 0.7904, 0.5393, 0.495, 0.7273,
     0.4149], [-0.1025, 0.2247, 0.0248, 0.8006, 0.5365, 0.5127, 0.7099, 0.4127], [-0.1046, 0.2248,
     0.028, 0.7984, 0.5296, 0.5301, 0.702, 0.4056]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Burgers64

```
[System-as-Prompt]
Dataset name: burgers_field64
System: burgers_field

[Task]
Estimate the 64-dimensional hidden state trajectory from the 64-dimensional noisy observation
     trajectory.
This is a 64-dimensional discretized Burgers field.
The observation is the full field with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.05. Observation noise standard deviation: 0.223607.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[0.1085, 0.3297, 0.1762, 0.5037, 0.6176, 0.8563, 0.8972, 0.9174], [0.1102, 0.3471, 0.175, 0.5038,
     0.5908, 0.8459, 0.9052, 0.9373], [0.098, 0.3405, 0.1937, 0.5134, 0.6197, 0.8695, 0.9238,
     0.9168], [0.0777, 0.3637, 0.188, 0.5079, 0.58, 0.8349, 0.9238, 0.9235], [0.1071, 0.3215,
     0.1935, 0.5065, 0.6225, 0.823, 0.8972, 0.946]]
States x[0:5] first 8 dimensions:
[[0.1116, 0.3305, 0.1844, 0.5154, 0.6148, 0.856, 0.9154, 0.9406], [0.1106, 0.3324, 0.1843, 0.511,
     0.6117, 0.8536, 0.9135, 0.9412], [0.1096, 0.3366, 0.1855, 0.5086, 0.6134, 0.8516, 0.9181,
     0.9411], [0.1039, 0.3361, 0.1894, 0.5058, 0.6109, 0.8485, 0.916, 0.9394], [0.1027, 0.339,
     0.1997, 0.5075, 0.6108, 0.8375, 0.9114, 0.9432]]
```

```
[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Fhn64

```
[System-as-Prompt]
Dataset name: fhn_lattice64
System: fhn_lattice

[Task]
Estimate the 64-dimensional hidden state trajectory from the 64-dimensional noisy observation
     trajectory.
This is a 64-dimensional FitzHugh-Nagumo reaction-diffusion lattice.
The observation is the full state with additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.08. Observation noise standard deviation: 0.282843.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[0.2844, 0.1067, 0.1804, 0.421, 0.3651, 0.3184, 0.474, 0.5171], [0.2379, -0.0369, 0.1219, 0.2634,
     0.3725, 0.5157, 0.4537, 0.5951], [0.2145, -0.0435, 0.1836, 0.3568, 0.3967, 0.5555, 0.6009,
     0.6249], [0.374, 0.2345, 0.2448, 0.437, 0.5156, 0.4965, 0.5294, 0.6889], [0.4028, 0.1505,
     0.2093, 0.5748, 0.4944, 0.6286, 0.753, 0.7599]]
States x[0:5] first 8 dimensions:
[[0.1994, 0.0263, 0.1689, 0.3331, 0.3279, 0.4625, 0.4332, 0.5183], [0.2372, 0.0497, 0.2047, 0.3893,
      0.3733, 0.5133, 0.4674, 0.575], [0.2805, 0.0906, 0.2314, 0.4449, 0.4169, 0.563, 0.5132,
     0.6331], [0.327, 0.1277, 0.2762, 0.5032, 0.4536, 0.6075, 0.5747, 0.701], [0.3714, 0.1604,
     0.3056, 0.5726, 0.4927, 0.6587, 0.6347, 0.7667]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```

Cubic-Sensoring100

```
[System-as-Prompt]
Dataset name: cubic_sensor100
System: cubic_sensor100

[Task]
Estimate the 100-dimensional hidden state trajectory from the 100-dimensional noisy observation
     trajectory.
This is a 100-dimensional full-observation cubic sensing filtering system.
The observation is the full state transformed by the dataset's cubic sensing construction plus
     additive Gaussian noise.

[Data]
Trajectory length: 200. Number of trajectories in this dataset file: 500.
Observation noise variance: 0.01. Observation noise standard deviation: 0.1.
The train split uses all trajectories except the last 20; the last 20 trajectories are used for
     validation and testing.

[Trajectory Example]
Observations y[0:5] first 8 dimensions:
[[-2.1607, -0.1955, -1.2797, 0.0794, 0.3535, -0.1437, 0.5383, -0.1237], [-1.8, -0.0597, -2.0251,
     0.0095, 0.8828, -0.3188, 0.3457, 0.2287], [-2.3851, -0.0745, -2.4508, -0.026, 0.9243, -0.5493,
      0.5109, 0.1766], [-1.7285, 0.0748, -2.7856, -0.0973, 0.7753, -0.6358, 0.3614, 0.2213],
     [-1.9571, -0.1297, -2.8405, 0.1241, 0.7282, -0.6965, 0.209, 0.4488]]
States x[0:5] first 8 dimensions:
[[-1.2634, -0.365, -1.0853, -0.0274, 0.692, -0.6119, 0.7662, 0.5761], [-1.254, -0.285, -1.2541,
     0.0712, 0.9095, -0.6892, 0.7398, 0.6089], [-1.3487, -0.3509, -1.3113, 0.0806, 0.9474, -0.873,
     0.823, 0.5561], [-1.1964, -0.3718, -1.374, 0.0336, 0.8959, -0.8565, 0.7579, 0.6208], [-1.2509,
      -0.5661, -1.4094, 0.0646, 0.8031, -0.9035, 0.597, 0.7041]]

[Filtering Goal]
Use the observation sequence and this system description to denoise the measurements and recover
     the state sequence.
```