# One Filters All: A Generalist Filter for State Estimation

**Shiqi Liu**[1 *] **Wenhan Cao**[1 *] **Chang Liu**[2] **Zeyu He**[1] **Tianyi Zhang**[1]
**Yinuo Wang**[1] **Shengbo Eben Li**[1 3 †]

[1] School of Vehicle and Mobility, Tsinghua University
[2] College of Engineering, Peking University [3] College of AI, Tsinghua University
[*]Equal contribution [†]Corresponding author

## Abstract

Estimating hidden states in dynamical systems, also known as optimal filtering, is a long-standing problem in various fields of science and engineering. In this paper, we introduce a general filtering framework, **LLM-Filter**[1], which leverages large language models (LLMs) for state estimation by embedding noisy observations with text prototypes. In various experiments for classical dynamical systems, we find that first, state estimation can significantly benefit from the reasoning knowledge embedded in pre-trained LLMs. By achieving proper modality alignment with the frozen LLM, LLM-Filter outperforms the state-of-the-art learning-based approaches. Second, we carefully design the prompt structure, System-as-Prompt (SaP), incorporating task instructions that enable the LLM to understand the estimation tasks. Guided by these prompts, LLM-Filter exhibits exceptional generalization, capable of performing filtering tasks accurately in changed or even unseen environments. We further observe a scaling-law behavior in LLM-Filter, where accuracy improves with larger model sizes and longer training times. These findings make LLM-Filter a promising foundation model of filtering.

## 1 Introduction

State estimation of dynamical systems is a crucial topic in various fields, including robotics [1], meteorology [2], chemistry [3], and transportation [4]. The most comprehensive framework for state estimation is Bayesian filtering [5], which performs online estimation by iteratively applying prediction and update steps. Popular online Bayes filters can be categorized into Gaussian filters [6] and particle filters (PFs) [7]. In high-dimensional non-Gaussian systems, Gaussian filters tend to produce significant errors [8, 9], whereas PFs are limited by their substantial computational demands [10, 11].

To address these limitations, there is a growing trend toward learning-based filtering methods [12, 13, 14, 15]. Trained offline on well-curated datasets, these learning-based filters directly capture accurate system statistics rather than relying on manual modeling, enabling high accuracy and efficient online estimation. However, these filters are typically tailored to specific tasks and require retraining when applied to different dynamical systems, as illustrated in Figure 1. Without retraining, their performance often deteriorates when the system undergoes changes or transitions to entirely new environments.

The control domain faces analogous challenges when adapting to diverse tasks and environments [16, 17]. Large control models have been proposed to address these challenges, with examples

[1]The code and dataset are publicly accessible at https://github.com/Anonymous-User-2367/LLM-Filter.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

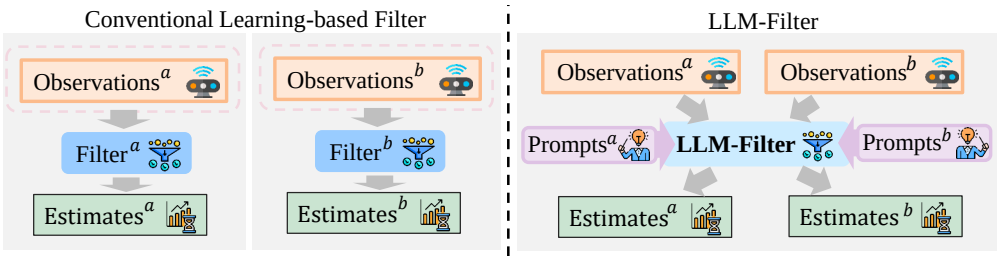



Figure 1: **Comparisons of LLM-Filter with Conventional Learning based Filters.** The left subfigure shows conventional learning-based filters, which rely on specific system data and training. The right subfigure shows LLM-Filter, which generalizes across different systems through prompt guidance, without requiring retraining.

like RT [18, 19] and OpenVLA [20]. By leveraging the large-scale pretraining knowledge of large language models (LLMs) and vision-language models (VLMs), these large control models demonstrate effective generalization across diverse robots, tasks, and environments [21]. It is worth noting that, *control is intrinsically a dual problem of filtering*, first demonstrated in linear systems through Lagrangian duality [22, 23], and later extended to nonlinear systems [24, 25, 26]. In the filtering domain, to the best of the authors' knowledge, no filtering method has yet integrated LLMs to utilize pre-training knowledge, nor has a general filtering model been developed.

To fill this research gap, we propose LLM-Filter, a generalist filter that reprograms LLMs for state estimation while preserving the integrity of the backbone model. The process begins by embedding noisy observations as text prototypes, which are subsequently processed by a frozen LLM to harmonize disparate data modalities. To enable generalization cross systems, we augment inputs with *System-as-Prompt* (SaP), which includes task-specific instructions and task examples. The outputs generated by the LLM are then mapped onto the final state estimates. By leveraging the pretraining knowledge of the LLM and guiding it through carefully designed prompts, LLM-Filter can outperform most state-of-the-art learning-based filters and exhibit remarkable in-context generalization, even when applied to completely new systems, as demonstrated in our experiments. Our main contributions are as follows:

- We demonstrate that state estimation can significantly benefit from the knowledge embedded in pre-trained language models. By establishing proper modality alignment between the filtering process and the inference mechanism of LLMs, we propose a generalist filter, called LLM-Filter.
- We carefully design an in-context approach, SaP, to help the LLM understand the current task and adapt to the specific application system. Guided by SaP, LLM-Filter exhibits exceptional generalization, effectively performing filtering tasks in unseen scenarios.
- In our experiments, LLM-Filter consistently outperforms the state-of-the-art learning-based filters and demonstrates remarkable in-context generalization in changed or entirely new systems without retraining or fine-tuning. Additionally, we observe a scaling-law behavior, where accuracy improves with larger model sizes and longer training times.

## 2 Preliminaries

### 2.1 Formulation of State Estimation

We begin with a brief review of state estimation and the primary filtering methods considered in this work. Typically, state estimation is modeled using a Markovian state-space model:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + \boldsymbol{\xi}_t, \tag{1a}$$
$$\boldsymbol{y}_t = h(\boldsymbol{x}_t) + \boldsymbol{\zeta}_t, \tag{1b}$$

for $t \in \{1, 2, \dots\}$. Here $f(\cdot)$ and $h(\cdot)$ represent the state transition and observation models, respectively. The term $\boldsymbol{x}_t \in \mathbb{R}^M$ is defined as the state while $\boldsymbol{y}_t \in \mathbb{R}^N$ denotes the observations. Furthermore, $\boldsymbol{\xi}_t$ and $\boldsymbol{\zeta}_t$ represent the transition noise and observation noise, respectively, with their

probability distributions generally assumed to be known. This state-space model description in (1) can be represented as a hidden Markov model:

$$\begin{aligned} \boldsymbol{x}_0 &\sim p(\boldsymbol{x}_0), \\ \boldsymbol{x}_t &\sim p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \\ \boldsymbol{y}_t &\sim p(\boldsymbol{y}_t|\boldsymbol{x}_t), \end{aligned} \tag{2}$$

where, $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ and $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ are the transition and output probabilities, respectively, and $p(\boldsymbol{x}_0)$ denotes the initial state distribution. Essentially, both (1) and (2) provide different formulations of the same underlying system model. The objective of state estimation is to extract the information of the state $\boldsymbol{x}_t$ on all available observations $\boldsymbol{y}_{1:t}$.

### 2.2 Online Bayesian Filtering

A principled framework for state estimation is Bayesian filtering, which computes $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ recursively through two steps:

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_{t-1}, \tag{3a}$$

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\mathrm{d}\boldsymbol{x}_t}, \tag{3b}$$

where, (3a) is called prediction step while (3b) is called the update step. Under the assumptions of linearity and Gaussian noise, (3) can be analytically calculated online, which is the basis of Kalman filter (KF) [27], as detailed in Appendix A. However, these filtering methods are highly dependent on accurate modeling $f(\cdot)$ and $h(\cdot)$, as well as the noise distributions of $\boldsymbol{\xi}_t$ and $\boldsymbol{\zeta}_t$ specific to each system.

### 2.3 Learning-based Filtering

Instead of performing online computations as in (3), learning-based filtering methods are trained on well-collected data to learn the correlation between observations and the underlying state. This learning is achieved by optimizing the following supervised learning loss to update the network parameters $\boldsymbol{\theta}$:

$$\mathcal{L}(\boldsymbol{\theta}) = ||\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t(\boldsymbol{y}_{1:t}, \boldsymbol{\theta})||_2^2, \tag{4}$$

where $\hat{\boldsymbol{x}}_t(\boldsymbol{y}_{1:t}, \boldsymbol{\theta})$ represents the estimates, and $\boldsymbol{x}_t$ is the ground truth. By capturing accurate system statistics from training data instead of relying on manual modeling, learning-based filtering methods often provide more precise estimates and greater efficiency. However, the heavy reliance on task-specific datasets renders conventional learning-based filters inflexible, requiring retraining whenever tasks change or transition to entirely new environments. To overcome this limitation, this paper aims to propose a general filtering method that leverages an in-context mechanism, enabling effective performance across multiple systems without retraining.

## 3 Methodology

The main structure of LLM-Filter is illustrated in Figure 2. We first align the modalities between the discrete tokens of the LLM and continuous observations (Section.3.1), then design an in-context inference approach based on system information to enhance generalization (Section.3.2), finally obtain the estimates from the predicted tokens of the LLM and define the loss function (Section.3.3).

### 3.1 Observation Embedding

**Moving Horizon.** As described in Section.2.1, the objective is to estimate the state $\boldsymbol{x}_t$ on all available observations $\boldsymbol{y}_{1:t}$. However, it is impractical to input an ever-growing sequence of observations. Inspired by the moving horizon estimation [28] and the context window of LLMs [29] , we design a sliding window with a fixed length $T$ for estimation. The input observations are defined as $\boldsymbol{Y}_t = \{\boldsymbol{y}_{t-T+1:t}\} \in \mathbb{R}^{T\times N}$ , and the corresponding output estimates are $\hat{\boldsymbol{X}}_t = \{\hat{\boldsymbol{x}}_{t:t+T-1}\} \in \mathbb{R}^{T\times M}$ with $\boldsymbol{X}_t = \{\boldsymbol{x}_{t:t+T-1}\} \in \mathbb{R}^{T\times M}$ representing the ground truth states, where $N$ and $M$ denote the dimensions of the observation and state variables, respectively.

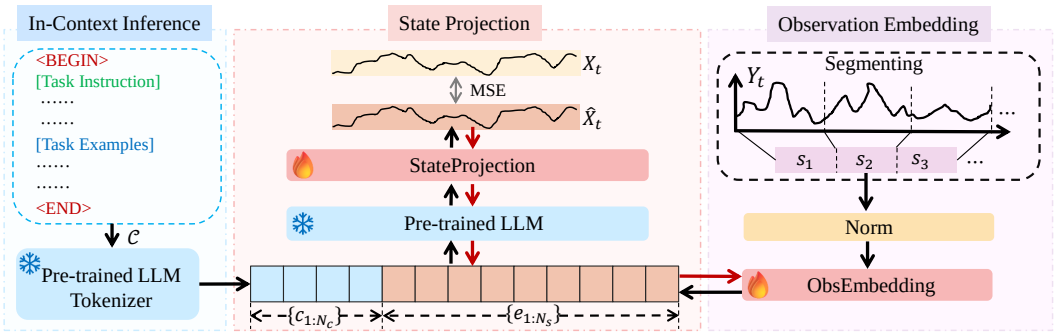



Figure 2: **Framework of LLM-Filter**. (1) **Observation Embedding:** Observations are segmented and embedded into token embeddings. (2) **In-context Inference:** In-context tokens from the tokenized SaP context are concatenated with the observation embeddings. (3) **State Projection:** The LLM's predicted tokens are projected into the state space to obtain the final estimates.

**Segment and Embedding.** To align the continuous observations with the discrete tokens, we segment the input observations and embed them into LLMs. A popular segmentation method uses a *single-series sequence* format [30, 31], which involves directly flattening all data dimensions. However, this approach can **disrupt the inherent correlations between variables**, such as the crucial relationship between position and speed, which is vital for accurate position estimation. To preserve the inherent correlations, we embed $\boldsymbol{Y}_t$ by segmenting it based on segment length $L$ while preserving its multi-dimensional structure:

$$\boldsymbol{Y}_t = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \cdots, \boldsymbol{s}_{N_s}\}, \quad N_s = \left\lfloor \frac{T}{L} \right\rfloor,$$

where each segment $\boldsymbol{s}_i \in \mathbb{R}^{L\times N}$ (for $i = 1, 2, \ldots, N_s$) and $\lfloor\cdot\rfloor$ denotes the floor function. If the division is not exact, padding is applied at the end to ensure all segments have the same length $L$. To leverage the pretraining knowledge and the intrinsic token transitions of the LLM, we freeze the parameters and eliminate the embedding and projection layers designed for language tokens. Instead, we introduce $\mathrm{ObsEmbedding}(\cdot) : \mathbb{R}^{L\times N} \mapsto \mathbb{R}^{D\times N}$, which independently embeds each normalized segment into the $D$-dimensional latent space of the LLM:

$$\boldsymbol{e}_i = \mathrm{ObsEmbedding}(\mathrm{Norm}(\boldsymbol{s}_i)), \quad i = 1, 2, \ldots, N_s.$$

### 3.2 In-context Inference

Traditional learning-based filters estimate the true system states solely from observational data, as discussed in Section 2.3. However, this reliance limits their ability to identify underlying system dynamics, resulting in poor generalization to new systems. To address this, we build upon the in-context learning capabilities of LLMs [33] and introduce a novel prompting strategy called SaP, which guides LLM-Filter to adapt flexibly to different systems. SaP is grounded in two components necessary for crafting effective prompts for LLM-Filter: Task Instruction and Task Examples. **Task Instruction** equips LLM-Filter with crucial contextual knowledge, which may differ across various domains. **Task Examples** guide LLM-Filter by showcasing example scenarios that clarify the task at hand. An example depiction of SaP can be found in Figure 3. When inference, the SaP text $\mathcal{C}$ will be fed to the pre-trained LLM tokenizer for processing language text:

<BEGIN>
[Task Instruction]
This task estimates the next state from a sequence of observations, where each observation represents the system's input states over time.
[Task Examples]
Input Observations (First 5):
[[-1.4645, 1.9252],[-1.4734, 1.9189],[-1.4821, 1.9124],[-1.4905, 1.9059],[-1.4988, 1.8993]]
True Output States (First 5):
[[-1.3500 0.6519],[-1.3646 0.6308],[-1.379, 0.6094],[-1.3931 0.5878], [-1.4069 0.5659]]
<END>

Figure 3: An Illustration of SaP for Hopf [32] system .

$$\{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_{N_c}\} = \mathrm{LLMTokenizer}(\mathcal{C}),$$

where $N_c$ is the number of tokenized context tokens.

### 3.3 State Projection

**Token Prediction.** Prevalent LLMs, trained on diverse downstream tasks, excel at predicting the next token based on previous ones across various domains [34, 35]. By inputting the guiding SaP and the embedding token of a previous observation, we aim to derive state estimation features for the next period. To achieve modality alignment, we utilize only the core layers of the LLM by excluding the original embedding and projection layers for language tokens. The tokenized SaP context and observation embeddings $\{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_{N_c}, \boldsymbol{e}_1, \cdots, \boldsymbol{e}_{N_s}\}$ are directly fed into the core layers, which then generate the corresponding output token embeddings $\{\hat{\boldsymbol{e}}_1, \cdots, \hat{\boldsymbol{e}}_{N_s}\}$:

$$\begin{aligned} \hat{E} &= \text{LLMCoreLayers}\left(\{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_{N_c}, \boldsymbol{e}_1, \cdots, \boldsymbol{e}_{N_s}\}\right), \\ \{\hat{\boldsymbol{e}}_1, \cdots, \hat{\boldsymbol{e}}_{N_s}\} &= \hat{E}_{\{N_c+1:N_c+N_s\}}. \end{aligned} \tag{5}$$

**Output Projection.** Afterward, we employ the layer $\text{StateProjection}(\cdot) : \mathbb{R}^{D\times N} \mapsto \mathbb{R}^{T\times M}$ to project the output embeddings $\{\hat{\boldsymbol{e}}_1, \cdots, \hat{\boldsymbol{e}}_{N_s}\}$ onto the system state space, thereby obtaining the final estimates $\hat{\boldsymbol{X}}_t$:

$$\hat{\boldsymbol{X}}_t = \text{StateProjection}(\{\hat{\boldsymbol{e}}_1, \cdots, \hat{\boldsymbol{e}}_{N_s}\}),$$

where both $\text{ObsEmbedding}(\cdot)$ and $\text{StateProjection}(\cdot)$ are implemented using simple MLPs, which has been shown to effectively embed and project time series for LLM [30].

**Loss Function.** Finally, the overall objective is to minimize the error between the ground truth $\boldsymbol{X}_t$ and the output estimations $\hat{\boldsymbol{X}}_t$ to optimize the parameters $\boldsymbol{\theta}$ of LLM-Filter $\mathcal{F}_{\boldsymbol{\theta}}$:

$$\mathcal{L}_{\boldsymbol{\theta}} = \frac{1}{T}||\boldsymbol{X}_t - \hat{\boldsymbol{X}}_t||_2^2 = \frac{1}{T}||\boldsymbol{X}_t - \mathcal{F}_{\boldsymbol{\theta}}(\boldsymbol{Y}_t, \mathcal{C})||_2^2.$$

This loss function can be seen as an extension of (4), incorporating additional context input. Notably, to preserve the pre-trained knowledge and due to resource constraints, we freeze the LLM and only update the parameters of $\text{ObsEmbedding}(\cdot)$ and $\text{StateProjection}$. The detailed configuration of LLM-Filter is provided in Appendix D.2.

*Remark* 1. When the SaP prompt is not available, LLM-Filter can still operate without in-context guidance, a variant we refer to as **LLM-Filter-O**. However, due to the lack of contextual guidance, its filtering accuracy and generalization capability may degrade, as demonstrated in the comparison provided in Section 5.

## 4 Related Work

**Online Bayesian Filtering.** The field of state estimation can be traced back to the pioneering development of the KF [27], which was famously applied to NASA's Apollo guidance and navigation systems at the Ames Research Center [36]. This milestone led to advanced methods like the extended Kalman filter (EKF) [37], unscented Kalman filter (UKF) [38], enabling applications in nonlinear systems. However, their reliance on linearization and Gaussian assumptions can introduce significant errors in complex scenarios. In response, particle-based filtering methods were developed, including the PF [39] and the ensemble Kalman filter (EnKF) [40]. Unfortunately, particle-based approaches face high computational demands and the curse of dimensionality.

**Learning-based Filtering.** Recent advances in neural network integration have driven the development of learning-based filtering methods. For instance, the KalmanNet [13, 41] adopts RNN to compute Kalman gains, significantly improving computational efficiency and accuracy. Similarly, MEstimator [14] and RStateNet [15] utilize multilayer perceptrons (MLPs) and long short-term memory networks (LSTMs) [42] to process historical state ensembles, enriching input information and achieving higher precision. ProTran [43], on the other hand, employs attention mechanisms to model system dynamics in the latent space, enabling long-term estimations. However, these methods rely solely on observational data as input and require retraining when applied to new systems. In contrast, our proposed LLM-Filter aligns the modalities of language and state estimation, allowing it to generalize to unseen systems through in-context guidance without additional training.

**Large Control Model.** As the dual problem of state estimation, there is an increasing trend toward training multi-task generalist control models on large, diverse datasets across different embodiments.

Recent work has incorporated LLMs and VLMs into control [16, 34], focusing primarily on high-level planning while underutilizing the knowledge from large-scale models. To address this, the RT model [18, 19, 21] directly train VLMs designed for open-vocabulary visual question answering to output low-level robot actions. Building on this architecture, OpenVLA [20] streamlines the model's parameters while improving performance and exploring further fine-tuning strategies for generalization. The rapid advancement of large control models drives our ambition to develop a generalist state estimation model that leverages pre-trained LLMs' knowledge.

**LLMs for Time Series Forecasting.** State estimation and time series forecasting both use sequential data. However, state estimation estimates hidden states from noisy observations, while time series forecasting predicts future values from the same data. Recent studies have made significant breakthroughs, especially in zero-shot forecasting, by establishing mappings between LLM tokens and numerical data [44, 45]. Additionally, TimeLLM [46]has explored the in-context forecasting capabilities of LLMs, leveraging an in-context learning to introduce a variety of forecasting tasks. AutoTimes [30] demonstrates how timestamp encoding techniques can unlock further potential for LLMs in time-series applications. While these models primarily predict future values based on past data, LLM-Filter focuses on estimating hidden states from noisy observations in dynamic systems.

## 5 Experiments

The goal of our experimental evaluations is to answer the following questions:

1. Can LLM-Filter learn to filter in canonical estimation task? (Section.5.1)
2. Can LLM-Filter effectively generalize to new systems? (Section.5.2)
3. Do the inherent characteristics of LLMs manifest within LLM-Filter? (Section.5.3)

**Benchmarks.** To comprehensively evaluate the performance of LLM-Filter, we conducted experiments on two types of classical dynamical systems: (1) low-dimensional nonlinear systems, and (2) high-dimensional chaotic systems, which are commonly used as benchmarks in previous studies [47, 48, 2]. Specifically, the low-dimensional nonlinear systems includes Selkov [49], Oscillator [50], Hopf [32], and Double Pendulum [51] systems. For high-dimensional chaotic systems, we performed experiments on Lorenz96 [52] and VL20 [53], both of which are widely used in atmospheric dynamics research. For the generalization experiments, we also include the Tracking system [5]. Detailed descriptions and configurations of each system are outlined in Appendix B. We utilize root mean square error (RMSE) and average runtime (ms/step) as the primary evaluation metrics, with their precise definitions provided in Appendix D.3.

**Baselines.** We compare LLM-Filter with state-of-the-art methods, including online Bayes filters: the PF [10], the EnKF [40], the average-particle ensemble Kalman filter (EnKFI) [48], the soft ensemble Kalman filter (EnKFS) [48], and the huber ensemble Kalman filter (HubEnKF) [54] as well as the learning-based methods: MEstimator [14], RStateNet [15], ProTran [43], and KalmanNet [13]. For a fair comparison, the hyperparameters of all online Bayes filters are selected during the first trial using the Bayesian Optimization (BO) package [55]. Detailed descriptions of these methods are provided in Appendix D.1. Unless otherwise stated, LLaMA-7B [56] is used as the base LLM in LLM-Filter.

### 5.1 Results for Canonical Estimation Task

In this experiment, we evaluate the fundamental filtering capabilities of LLM-Filter and LLM-Filter-O on canonical estimation tasks, comparing their performance to state-of-the-art methods including MEstimator, RStateNet, ProTran, KalmanNet, EnKF, and PF.

**Low-dimensional Nonlinear Systems.** We first conducted experiments on low-dimensional nonlinear systems. A summary of the results is presented in Table 1, where LLM-Filter consistently outperform all other methods across all tested systems. Notably, the RMSE of the LLM-Filter shows a maximum reduction of $32.00\%$ across all systems. On average, it demonstrates an improvement of $17.93\%$ over the best-performing learning-based filtering methods. When compared to the top-performing online Bayes filter, LLM-Filter still achieves an average improvement of $21.65\%$. In addition, with the use of the in-context prompt, LLM-Filter achieves a $10.18\%$ improvement over the data-only LLM-Filter-O, indicating that in-context prompting helps LLM-Filter better understand the system

and perform more accurate state estimation. These results highlight the strong filtering capabilities of the LLM-Filter in low-dimensional nonlinear systems, establishing its state-of-the-art performance.

Table 1: RMSE of State Estimation on Classical Systems. The best performances are marked in **bold**, and the second-best performances are underlined. Results labeled as *NaN* indicate cases where the models diverged during training.

| Method | Learning-Based Filters | | | | | | Online Bayes Filters | |
|---|---|---|---|---|---|---|---|---|
| | LLM-Filter | LLM-Filter-O | MEstimator | RStateNet | ProTran | KalmanNet | EnKF | PF |
| **Selkov** | **0.4061** | 0.6369 | 0.8864 | 0.7202 | 1.0219 | 1.1662 | 0.5978 | 0.6863 |
| **Oscillator** | **0.5247** | 0.5753 | 0.8347 | 0.8493 | 0.8933 | 0.5665 | 0.5505 | 0.6807 |
| **Hopf** | **0.5751** | 0.8180 | 0.8290 | 0.7282 | 0.7146 | 1.1984 | 0.6322 | 0.6801 |
| **Pendulum** | **0.8348** | 0.9218 | 0.9354 | 0.9180 | 0.8456 | 2.7140 | 1.4117 | 5.3788 |
| **Lorenz96** | **0.9149** | 0.9735 | 0.9649 | 0.9762 | 0.9975 | NaN | 6.6024 | 4.6289 |
| **VL20** | **0.7717** | 0.8433 | 1.0014 | 0.9428 | 0.9902 | NaN | 5.8633 | 11.6980 |
| **Average** | **0.6712** | 0.7948 | 0.9086 | 0.8558 | 0.9105 | NaN | 2.6097 | 3.9588 |

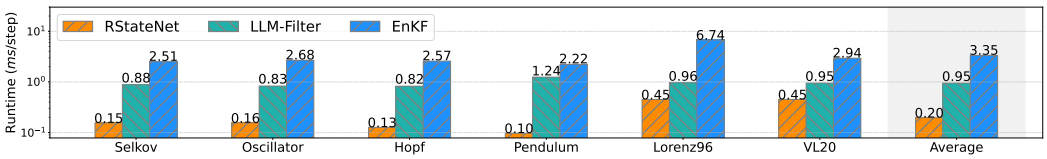



Figure 4: Comparison of Estimation Runtime Between LLM-Filter, EnKF, and RStateNet. Comprehensive results are presented in Tab 6.

**High-dimensional Chaotic Systems.** We then performed experiments on the Lorenz96 and VL20 systems, both with 72 state dimensions, designed to simulate atmospheric dynamics. The results are summarized in Table 1. LLM-Filter achieves state-of-the-art performance on the Lorenz96 and VL20 systems, with an average improvement of $11.7\%$ over the best baseline filters. In contrast, the KalmanNet, as implemented in the official codebase[2], ultimately suffers from divergence during training. These results validate that LLM-Filter is capable of effectively addressing high-dimensional chaotic systems.

**Running Time.** The running time is a crucial factor in practical filtering applications. For learning-based filters and online Bayes filters, we selected RStateNet and EnKF, respectively, as representative methods for comparison. The results are summarized in Figure 4. For all systems, the running time of the LLM-Filter is slower than RStateNet, which has fewer parameters. However, LLM-Filter is faster than the online EnKF, making it a highly practical and effective solution for real-time applications. As the system dimension increases, both EnKF and RStateNet show a significant increase in computation time. In contrast, LLM-Filter maintains a stable runtime of around $1.0ms$, which highlights the LLM-Filter's efficiency in high-dimensional system estimation, offering a clear advantage for such applications. Detailed information about the testing equipment configuration, model parameters, and peak memory can be found in Appendix D and Table 5.

### 5.2 Generalization Evaluation for Estimation Task

In this experiment, we assess the cross-adaptation capability of our proposed LLM-Filter—specifically, its performance when applied to a modified system or even an entirely different system.

**Model Mismatch.** Model mismatch, a common challenge in filtering tasks, occurs when the assumed observation noise variance differs from the actual noise variance [13, 48]. We evaluated LLM-Filter under model mismatch on the Selkov, Oscillator, and Hopf systems, using the observation covariance expansion ratio (OCER) to quantify mismatch severity. We compared LLM-Filter with the robust filters designed for model mismatch, including EnKFI, EnKFS, and HubEnKF. The results, shown in Figure 5. The performance of the robust filters rapidly declined as the OCER increased, while

[2]The official open-source repository is available at https://github.com/KalmanNet/KalmanNet_TSP. Despite optimizing the hyperparameters and network architecture, KalmanNet still struggles to filter high-dimensional systems, potentially due to the large dimensionality involved in computing the Kalman gain.

LLM-Filter maintained consistent accuracy with minimal degradation. These findings underscore the LLM-Filter's exceptional robustness and adaptability with varying observation conditions.

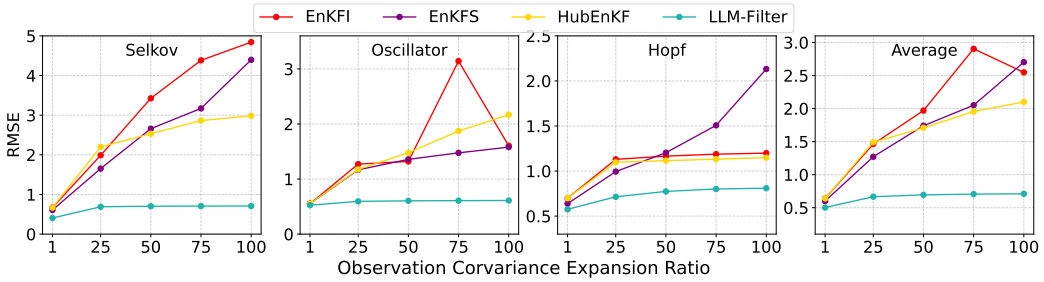



Figure 5: Comparison of RMSE Results across the Selkov, Oscillator, and Hopf systems as the observation covariance expansion ratio increases. The rightmost subfigure presents the average results across these systems. Complete results are presented in Table 7.

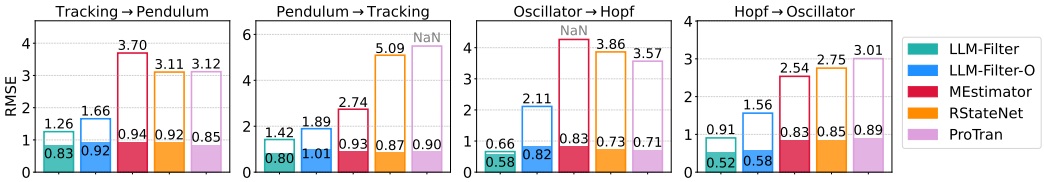



Figure 6: RMSE for Cross-System Scenarios. The *hollow bars* represent cross-system scenarios, while the *solid bars* correspond to cases where the model is both trained and tested on the same system. In cross-system scenarios, *Tracking* $\rightarrow$ *Pendulum* indicates that the model is trained on the Tracking system and evaluated on the Pendulum system. Comprehensive results are presented in Tab 9.

**Cross Systems.** In this experiment, we evaluate the performance on completely different systems, presented in Figure 6. The cross-system results are shown with *hollow bars*, while *solid bars* represent the performance of baseline models trained directly on the target systems. We also include the LLM-Filter-O as an ablation comparison. Guided by the SaP prompt, LLM-Filter achieves exceptional performance, demonstrating the lowest RMSE across all methods in cross-system scenarios. When compared with specific training baselines, LLM-Filter delivers competitive performance, even achieving the best results in the *Oscillator* $\rightarrow$ *Hopf* scenario. Without the guidance of the SaP prompt, LLM-Filter-O shows a noticeable decrease in its generalization capability compared to LLM-Filter. Moreover, the minimal difference between the solid and hollow bars for LLM-Filter indicates that it maintains strong performance even when applied to previously unseen systems. In contrast, the other learning-based methods experience a significant decline in performance, even losing their ability to provide accurate estimates when confronted with unseen systems. These results demonstrate the potential of LLM-Filter to generalize effectively across new systems.

### 5.3 Exploration of LLMs

In this subsection, we explore the scaling law and fine-tuning methods of LLMs for LLM-Filter. Additionally, we conduct an ablation study on the LLM component to emphasize its crucial role in achieving the model's success.

**Scaling Behavior.** Scaling laws represent empirical observations about how performance and efficiency metrics vary as model size increases [57]. In our study, we explore the scaling behavior of LLM-Filter in state estimation tasks by evaluating models of different parameter sizes as the LLM backbone. To avoid inconsistencies caused by varying tokenizers across models, we did not use in-context prompting in this experiment. We examined the average RMSE and training efficiency across all low-dimensional nonlinear systems and high-dimensional chaotic systems, with results illustrated in Figure 7. Our findings illustrate the scaling behavior of LLM-Filter: as model parameters increase, RMSE decreases, boosting estimation accuracy, though at the cost of longer

Table 2: RMSE for LLM-Filter Using LoRA or Full Fine-Tuning.

| Method | LLM-Filter | +LoRA | +Full Finetune |
|---|---|---|---|
| **Selkov** | **0.4061** | 0.4498 | 0.7103 |
| **Oscillator** | **0.5247** | 0.5370 | 0.5513 |
| **Hopf** | **0.5751** | 0.6744 | 0.8444 |
| **Lorenz96** | 0.9149 | 0.9147 | **0.9140** |
| **VL20** | 0.7717 | 0.8962 | **0.7027** |

Table 3: RMSE for LLM Ablation Study of LLM-Filter.

| Method | LLM-Filter | MLP | RNN | Transformer |
|---|---|---|---|---|
| **Selkov** | **0.4061** | 0.9910 | 0.9862 | 0.8863 |
| **Oscillator** | **0.5247** | 0.8176 | 0.9481 | 0.8566 |
| **Hopf** | **0.5751** | 0.9057 | 1.0114 | 0.8610 |
| **Lorenz96** | **0.9149** | 0.9661 | 1.0105 | 0.9721 |
| **VL20** | **0.7717** | 0.9977 | 0.8043 | 0.9598 |

training times. Furthermore, our experiments underscore the adaptability of LLM-Filter, as it delivers strong performance across a variety of LLM bases.

**Full Fine-Tuning and LoRA.** This experiment investigates two prominent methods commonly used to adapt LLMs more effectively for downstream tasks: full parameter fine-tuning [58] and low-rank adaptation (LoRA) [59]. For LLM-Filter, full fine-tuning involves updating all model parameters, whereas LoRA only updates a low-rank matrix within the model. A summary of the results is shown in Table 2. Interestingly, equipping LLM-Filter with LoRA led to a decline in performance. We attribute this to the limitations of low-rank matrix adjustments, which may lack the expressive power needed for the high precision required in state estimation tasks. On the other hand, a fully fine-tuned LLM-Filter demonstrated improved performance in more challenging environments, such as Lorenz96, and VL20 systems. This comes at a cost: in simpler dynamic systems, the performance was worse, likely due to overfitting from extensive parameter updates.

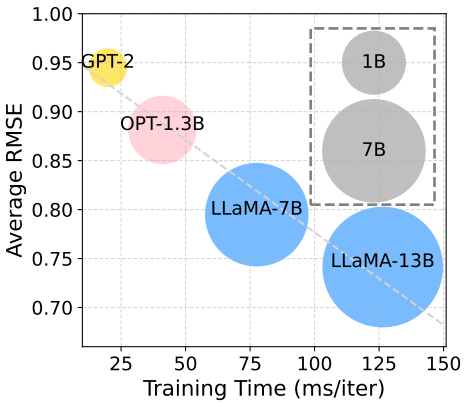



Figure 7: Comparison of RMSE and Efficiency Across Various LLMs. Complete results can be found in Table 10.

**LLM Ablation Study.** In this experiment, we address whether the use of LLMs is truly essential for state estimation tasks, a concern raised in recent studies on LLMs for time series analysis [60]. To investigate this, we replace the LLM in LLM-Filter with traditional neural network architectures, including MLP, RNN, and Transformer, while keeping all other components unchanged. The results are summarized in Table 3. Our findings clearly demonstrate that LLM-Filter consistently outperforms these traditional network architectures across all tested systems. Specifically, the RMSE of LLM-Filter shows an average improvement of $31.66\%$, $32.43\%$, and $30.73\%$ over MLP, RNN, and Transformer, respectively. This significant improvement underscores the superior performance and essential role of LLMs in state estimation tasks, validating their necessity for achieving state-of-the-art results.

## 6 Conclusion and Future Work

Inspired by the success of large control models, we propose a general large filtering model, LLM-Filter, designed to solve specific estimation tasks, including generalization tasks, by transferring knowledge from LLMs. The process begins by embedding noisy observations with text prototypes and then enriching the input with the SaP method, which incorporates basic task instructions and characteristics. These inputs are then processed by a frozen LLM. The generated outputs are projected onto the final estimates. Guided by prompts and leveraging pretraining knowledge, LLM-Filter outperforms specialized learning-based methods and demonstrates strong generalization across various systems.

The generalization ability of LLM-Filter is currently restricted to systems with the same dimensionality, and its evaluation across diverse settings is still limited. Future work will focus on training large-scale state estimation models with multi-source data, aiming to enable generalization across systems with varying dimensions. We hope that our contributions will lay the groundwork for addressing state estimation challenges in a wide range of real-world applications.

## Acknowledgements

This work was supported by the Tsinghua–Efort Joint Research Center for EAI Computation and Perception, and by the Anhui Province Science and Technology Innovation Plan under the project "Breakthrough and Industrialization of L4-Level Multi-Modal Perception End-to-End Model" (No. 202423d12050007).

## References

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

[2] Kevin Course and Prasanth B Nair. "State estimation of a physical system with unknown governing equations". In: *Nature* 622.7982 (2023), pp. 261–267.

[3] Denis Dochain. "State and parameter estimation in chemical and biochemical processes: a tutorial". In: *Journal of process control* 13.8 (2003), pp. 801–818.

[4] Jinlei Zhang et al. "Physics-informed deep learning for traffic state estimation based on the traffic flow model and computational graph method". In: *Information Fusion* 101 (2024), p. 101971.

[5] Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*. Vol. 17. Cambridge university press, 2023.

[6] Kazufumi Ito and Kaiqi Xiong. "Gaussian filters for nonlinear filtering problems". In: *IEEE transactions on automatic control* 45.5 (2000), pp. 910–927.

[7] Zhe Chen et al. "Bayesian filtering: From Kalman filters to particle filters, and beyond". In: *Statistics* 182.1 (2003), pp. 1–69.

[8] Hamed H Afshari, S Andrew Gadsden, and S Habibi. "Gaussian filters for parameter and state estimation: A general review of theory and recent trends". In: *Signal Processing* 135 (2017), pp. 218–238.

[9] Ondřej Straka, Jindřich Duník, and Victor Elvira. "Importance Gauss-Hermite Gaussian Filter for Models with Non-Additive Non-Gaussian Noises". In: *2021 IEEE 24th International Conference on Information Fusion (FUSION)*. IEEE. 2021, pp. 1–7.

[10] Peter Jan Van Leeuwen et al. "Particle filters for high-dimensional geoscience applications: A review". In: *Quarterly Journal of the Royal Meteorological Society* 145.723 (2019), pp. 2335–2365.

[11] Ziyu Wan and Lin Zhao. "DiffPF: Differentiable Particle Filtering with Generative Sampling via Conditional Diffusion Models". In: *arXiv preprint arXiv:2507.15716* (2025).

[12] Yujie Tang et al. "Reinforcement learning compensated extended Kalman filter for attitude estimation". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 6854–6859.

[13] Guy Revach et al. "KalmanNet: Neural network aided Kalman filtering for partially known dynamics". In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 1532–1547.

[14] Gwanghyeon Ji et al. "Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4630–4637.

[15] Pragyan Dahal et al. "RobustStateNet: Robust ego vehicle state estimation for Autonomous Driving". In: *Robotics and Autonomous Systems* 172 (2024), p. 104585.

[16] Michael Ahn et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *arXiv preprint arXiv:2204.01691* (2022).

[17] Octo Model Team et al. "Octo: An open-source generalist robot policy". In: *arXiv preprint arXiv:2405.12213* (2024).

[18] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[19] Anthony Brohan et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *arXiv preprint arXiv:2307.15818* (2023).

[20] Moo Jin Kim et al. “OpenVLA: An Open-Source Vision-Language-Action Model”. In: *arXiv preprint arXiv:2406.09246* (2024).

[21] Abby O’Neill et al. “Open x-embodiment: Robotic learning datasets and rt-x models”. In: *arXiv preprint arXiv:2310.08864* (2023).

[22] Thomas Kailath, Ali H Sayed, and Babak Hassibi. *Linear estimation*. Prentice Hall, 2000.

[23] Graham C Goodwin et al. “Lagrangian duality between constrained estimation and control”. In: *Automatica* 41.6 (2005), pp. 935–944.

[24] Emanuel Todorov. “General duality between optimal control and estimation”. In: *2008 47th IEEE conference on decision and control*. IEEE. 2008, pp. 4286–4292.

[25] Jin Won Kim. “Duality for nonlinear filtering”. In: *arXiv preprint arXiv:2207.07709* (2022).

[26] Yinuo Wang et al. “ODE-based Smoothing Neural Network for Reinforcement Learning Tasks”. In: *The Thirteenth International Conference on Learning Representations*. 2025.

[27] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960).

[28] Angelo Alessandri, Marco Baglietto, and Giorgio Battistelli. “Moving-horizon state estimation for nonlinear discrete-time systems: New stability results and approximation schemes”. In: *Automatica* 44.7 (2008), pp. 1753–1765.

[29] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[30] Yong Liu et al. “Autotimes: Autoregressive time series forecasters via large language models”. In: *arXiv preprint arXiv:2402.02370* (2024).

[31] Yong Liu et al. “Timer: Generative Pre-trained Transformers Are Large Time Series Models”. In: *Forty-first International Conference on Machine Learning*. 2024.

[32] Brian D Hassard, Nicholas D Kazarinoff, and Yieh-Hei Wan. *Theory and applications of Hopf bifurcation*. Vol. 41. CUP Archive, 1981.

[33] Shukang Yin et al. “A survey on multimodal large language models”. In: *arXiv preprint arXiv:2306.13549* (2023).

[34] Danny Driess et al. “Palm-e: An embodied multimodal language model”. In: *arXiv preprint arXiv:2303.03378* (2023).

[35] Jiaming Han et al. “Onellm: One framework to align all modalities with language”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 26584–26595.

[36] Mohinder S Grewal and Angus P Andrews. “Applications of Kalman filtering in aerospace 1960 to the present [historical perspectives]”. In: *IEEE Control Systems Magazine* 30.3 (2010), pp. 69–78.

[37] 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*. Vol. 135. National Aeronautics and Space Administration, 1962.

[38] Simon J Julier, Jeffrey K Uhlmann, and Hugh F Durrant-Whyte. “A new approach for filtering nonlinear systems”. In: *Proceedings of 1995 American Control Conference-ACC’95*. Vol. 3. IEEE. 1995, pp. 1628–1632.

[39] Jun S Liu and Rong Chen. “Sequential Monte Carlo methods for dynamic systems”. In: *Journal of the American statistical association* 93.443 (1998), pp. 1032–1044.

[40] Pavel Sakov and Peter R Oke. “Implications of the form of the ensemble transformation in the ensemble square root filters”. In: *Monthly Weather Review* 136.3 (2008), pp. 1042–1053.

[41] Xiaoyong Ni, Guy Revach, and Nir Shlezinger. “Adaptive Kalmannet: Data-Driven Kalman Filter with Fast Adaptation”. In: *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2024, pp. 5970–5974.

[42] S Hochreiter. “Long Short-term Memory”. In: *Neural Computation MIT-Press* (1997).

[43] Binh Tang and David S Matteson. “Probabilistic transformer for time series analysis”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 23592–23608.

[44] Tian Zhou et al. “One fits all: Power general time series analysis by pretrained lm”. In: *Advances in neural information processing systems* 36 (2023), pp. 43322–43355.

[45] Nate Gruver et al. “Large language models are zero-shot time series forecasters”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[46] Ming Jin et al. “Time-llm: Time series forecasting by reprogramming large language models”. In: *arXiv preprint arXiv:2310.01728* (2023).

[47] Patrick N Raanes, Yumeng Chen, and Colin Grudzien. “DAPPER: data assimilation with Python: a package for experimental research”. In: *Journal of Open Source Software* 9.94 (2024), p. 5150.

[48] Gerardo Duran-Martin et al. “Outlier-robust Kalman Filtering through Generalised Bayes”. In: *arXiv preprint arXiv:2405.05646* (2024).

[49] Evgeni E SEL’KOV. “Self-Oscillations in Glycolysis 1. A Simple Kinetic Model”. In: *European Journal of Biochemistry* 4.1 (1968), pp. 79–86.

[50] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[51] RB Levien and SM Tan. “Double pendulum: An experiment in chaos”. In: *American Journal of Physics* 61 (1993), pp. 1038–1038.

[52] Edward N Lorenz. “Predictability: A problem partly solved”. In: *Proc. Seminar on predictability*. Vol. 1. 1. Reading. 1996.

[53] Gabriele Vissio and Valerio Lucarini. “Mechanics and thermodynamics of a new minimal model of the atmosphere”. In: *The European Physical Journal Plus* 135.10 (2020), pp. 1–21.

[54] Soojin Roh et al. “Observation quality control with a robust ensemble Kalman filter”. In: *Monthly Weather Review* 141.12 (2013), pp. 4414–4428.

[55] Jacob R Gardner et al. “Bayesian optimization with inequality constraints.” In: *ICML*. Vol. 2014. 2014, pp. 937–945.

[56] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[57] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[58] Jesse Dodge et al. “Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping”. In: *arXiv preprint arXiv:2002.06305* (2020).

[59] Edward J Hu et al. “Lora: Low-rank adaptation of large language models”. In: *arXiv preprint arXiv:2106.09685* (2021).

[60] Mingtian Tan et al. “Are language models actually useful for time series forecasting?” In: *arXiv preprint arXiv:2406.16964* (2024).

[61] Alberto Carrassi et al. “Data assimilation in the geosciences: An overview of methods, issues, and perspectives”. In: *Wiley Interdisciplinary Reviews: Climate Change* 9.5 (2018), e535.

[62] John C Butcher. “Implicit runge-kutta processes”. In: *Mathematics of computation* 18.85 (1964), pp. 50–64.

[63] Xinshuo Weng et al. “Ab3dmot: A baseline for 3d multi-object tracking and new evaluation metrics”. In: *arXiv preprint arXiv:2008.08063* (2020).

[64] R OpenAI. “Gpt-4 technical report. arxiv 2303.08774”. In: *View in Article* 2.5 (2023).

[65] Guojian Zhan et al. “Bicriteria Policy Optimization for High-Accuracy Reinforcement Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems* 37.1 (2026), pp. 312–326.

[66] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[67] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.

[68] Ilya Loshchilov, Frank Hutter, et al. “Fixing weight decay regularization in adam”. In: *arXiv preprint arXiv:1711.05101* 5 (2017).

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: The limitations of our work are discussed in Section 6.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all necessary details to reproduce the main experimental results in Section 5.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our experimental data and code have been made publicly available and are accessible at https://github.com/Anonymous-User-2367/LLM-Filter.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all training and testing details, as well as hyperparameter analysis, in Appendix D and Appendix E.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper presents the RMSE results obtained from our experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Sufficient information on the computational resources is provided in Appendix D.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research presented in this paper fully conforms, in all respects, to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The societal impacts of this work are discussed in Section 6.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Yes, the creators and original owners of assets are properly credited, and their licenses and terms of use are explicitly mentioned and respected.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

# A Bayes Filters

In this section, we introduce the classical Bayes filters, including Kalman filter (KF), Extended Kalman filter (EKF), Ensemble Kalman filter (EnKF), and Particle filter (PF). For detailed descriptions of advanced variants, readers are encouraged to explore the relevant literature cited in Section 5.

## A.1 Kalman Filter

Suppose the SSM is linear and Gaussian:

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{F}_t \boldsymbol{x}_t + \boldsymbol{\xi}_t, \\ \boldsymbol{y}_t &= \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{\zeta}_t, \end{aligned}$$

where $\boldsymbol{\xi}_t \sim \mathcal{N}(\boldsymbol{\xi}_t|0, \boldsymbol{Q_t})$ and $\boldsymbol{\zeta}_t \sim \mathcal{N}(\boldsymbol{\zeta}_t|0, \boldsymbol{R_t})$ represent zero-mean Gaussian noise terms with covariance matrices $\boldsymbol{Q_t} \in \mathbb{R}^{M\times M}$ and $\boldsymbol{R_t} \in \mathbb{R}^{N\times N}$, respectively. The initial state is a Gaussian distribution, given by $\boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{x}_0|\boldsymbol{\mu}_0, \boldsymbol{\Sigma_0})$ with known mean $\boldsymbol{\mu}_0 \in \mathbb{R}^M$ and covariance $\boldsymbol{\Sigma_0} \in \mathbb{R}^{M\times M}$. According to the Kalman filter [27], the prior predictive distribution and posterior distribution remain Gaussian:

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) &= \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1}), \\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &= \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t). \end{aligned}$$

The mean and covariance of the prior predictive distribution are denoted by:

$$\begin{aligned} \boldsymbol{\mu}_{t|t-1} &= \boldsymbol{F}_t \boldsymbol{\mu}_{t-1}, \\ \boldsymbol{\Sigma}_{t|t-1} &= \boldsymbol{F}_t \boldsymbol{\Sigma}_{t-1} \boldsymbol{F}_t^\top + \boldsymbol{Q}_t, \end{aligned}$$

The posterior mean and covariance are given by:

$$\begin{aligned} \boldsymbol{\Sigma}_t^{-1} &= \boldsymbol{\Sigma}_{t|t-1}^{-1} + \boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1} \boldsymbol{H}_t, \\ \mathbf{K}_t &= \boldsymbol{\Sigma}_t \boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}, \\ \boldsymbol{\mu}_t &= \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \left( \boldsymbol{y}_t - \boldsymbol{H}_t \boldsymbol{\mu}_{t|t-1} \right). \end{aligned}$$

where $\boldsymbol{K}_t$ is the Kalman gain matrix.

## A.2 Extended Kalman Filter

When faced with nonlinear systems, EKF introduces the first-order Taylor approximations of the nonlinear transition model $f(\cdot)$ and observation model $h(\cdot)$. We denote the Jacobians of these functions as $\boldsymbol{F_x}(\cdot)$ and $\boldsymbol{H_x}(\cdot)$. Then the predict step of EKF is:

$$\begin{aligned} \boldsymbol{\mu}_{t|t-1} &= f(\boldsymbol{\mu}_{t-1}), \\ \boldsymbol{\Sigma}_{t|t-1} &= \boldsymbol{F}_x(\boldsymbol{\mu}_{t-1}) \boldsymbol{\Sigma}_{t-1} \boldsymbol{F}_x(\boldsymbol{\mu}_{t-1})^\top + \boldsymbol{Q}_t. \end{aligned}$$

The update step is

$$\begin{aligned} \boldsymbol{\Sigma}_t^{-1} &= \boldsymbol{\Sigma}_{t|t-1}^{-1} + \boldsymbol{H}_x(\boldsymbol{\mu}_{t|t-1})^\top \boldsymbol{R}_t^{-1} \boldsymbol{H}_x(\boldsymbol{\mu}_{t|t-1}), \\ \mathbf{K}_t &= \boldsymbol{\Sigma}_t \boldsymbol{H}_x(\boldsymbol{\mu}_{t|t-1})^\top \boldsymbol{R}_t^{-1}, \\ \boldsymbol{\mu}_t &= \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t \left( \boldsymbol{y}_t - \boldsymbol{H}_x(\boldsymbol{\mu}_{t|t-1}) \boldsymbol{\mu}_{t|t-1} \right). \end{aligned}$$

## A.3 Ensemble Kalman Filter

The EnKF was developed as an alternative to the EKF for high-dimensional and chaotic systems, such as weather forecasting [61]. Instead of explicitly computing the covariance matrix, EnKF represents the belief state with an ensemble of $N_e$ particles $\hat{\boldsymbol{x}}_t^i \in \mathbb{R}^M$ , where $i = 1, \cdots, N_e$. The predict step of EnKF obtains $\hat{\boldsymbol{x}}_{t|t-1}^i$ using the transition model from (3a). In the update step, predicted observations $\hat{\boldsymbol{y}}_t^i$ are sampled for each particle as:

$$\hat{\boldsymbol{y}}_t^i \sim \mathcal{N}(\hat{\boldsymbol{y}}_t^i | h(\hat{\boldsymbol{x}}_{t|t-1}^i), \boldsymbol{R}_t) \quad i = 1, 2, \cdots, N_e.$$

Next, the particles are updated as follows:

$$\boldsymbol{K}_t = \mathrm{Cov}(\hat{\boldsymbol{x}}^i_{t|t-1}, \hat{\boldsymbol{y}}^i_t)\mathbb{V}(\hat{\boldsymbol{y}}^i_t)^{-1},$$
$$\hat{\boldsymbol{x}}^i_t = \hat{\boldsymbol{x}}^i_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}^i_t),\ i = 1, 2, \cdots, N_e,$$

where $\mathrm{Cov}(\cdot,\cdot)$ and $\mathbb{V}(\cdot)$ denote the covariance and variance, respectively. Finally, the state estimate at time $t$ is calculated as the average of all particles:

$$\hat{\boldsymbol{x}}_t = \sum_{i=1}^{N_e} \boldsymbol{x}^i_t.$$

### A.4 Particle Filter

The PF is a filtering method based on sequential importance sampling [39], characterized by high precision but relatively low computational efficiency. The standard implementation of PF follows the bootstrap procedure outlined below:

(1) **Sampling:** Generate samples for all $N_p$ particles;

$$\boldsymbol{x}^i_t \sim p(\boldsymbol{x}_t|\boldsymbol{x}^i_{t-1}), \quad i = 1, 2, \cdots, N_p.$$

(2) **Weighting:** compute weights of all particles and normalize;

$$w^i_t \propto p(\boldsymbol{y}_t|\boldsymbol{x}^i_t), \quad i = 1, 2, \cdots, N_p.$$

(3) **Resampling**: Resample particles based on their normalized weights.

Finally, the state estimate at time $t$ is calculated as the weighted average of all particles:

$$\hat{\boldsymbol{x}}_t = \sum_{i=1}^{N_p} w^i_t \boldsymbol{x}^i_t.$$

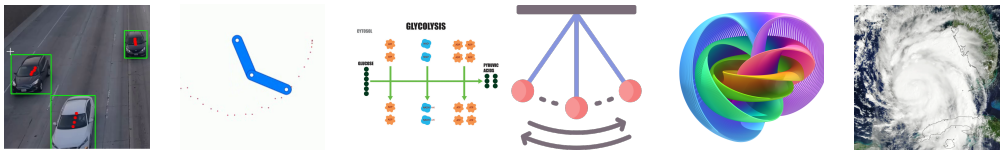
Table 4: System Specifications for Experiments. The dimensions of the state and observation spaces are denoted by $M$ and $N$, respectively. *Dataset Size* refers to the collected data from the system, specified as (trajectory length, number of trajectories, and state dimension). The *Forward Euler* method refers to a first-order explicit discretization scheme, while *RK4* represents the fourth-order Runge-Kutta discretization scheme [62].

| System | Linear | M | N | Dataset Size | $\boldsymbol{Q}$ | $\boldsymbol{R}$ | $\Delta t$ | Discretization | Application Domain |
|---|---|---|---|---|---|---|---|---|---|
| Tracking | T | 4 | 2 | (200, 100, 6) | $0.1\boldsymbol{I}$ | $10\boldsymbol{I}$ | 0.1 | Forward Euler | Target Tracking |
| Selkov | F | 2 | 2 | (200, 100, 4) | $\boldsymbol{I}$ | $\boldsymbol{I}$ | 0.01 | RK4 | Glycolysis Process |
| Oscillator | F | 2 | 2 | (200, 100, 4) | $\boldsymbol{I}$ | $\boldsymbol{I}$ | 0.01 | RK4 | Oscillatory Motion |
| Hopf | F | 2 | 2 | (200, 100, 4) | $\boldsymbol{I}$ | $\boldsymbol{I}$ | 0.01 | RK4 | Chemical Reactions |
| Pendulum | F | 4 | 2 | (200, 100, 6) | $\boldsymbol{I}$ | $\boldsymbol{I}$ | 0.01 | RK4 | Physical Dynamics |
| Lorenz 96 | F | 72 | 72 | (200, 100, 144) | $\boldsymbol{I}$ | $100\boldsymbol{I}$ | 0.01 | RK4 | Atmospheric Dynamics |
| VL20 | F | 72 | 72 | (200, 100, 144) | $\boldsymbol{I}$ | $\boldsymbol{I}$ | 0.01 | RK4 | Atmospheric Dynamics |

## B System Descriptions

The systems used in this experiment are based on real-world models that are continuous in nature and governed by ordinary differential equations (ODEs). To map these continuous systems into the corresponding Markovian state-space model (SMM), we discretize them using methods such as the forward Euler method and the fourth-order Runge-Kutta discretization [62]. For the uncertain noise terms in (1), denoted as $\boldsymbol{\xi}_t$ and $\boldsymbol{\zeta}_t$, we assume they follow a standard Gaussian distribution, ensuring consistency across the models.

$$\boldsymbol{\xi}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}),\ \boldsymbol{\zeta}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}),$$




(a) (b) (c) (d) (e) (f)

Figure 8: Illustration of Experiment Systems. (a) Tracking; (b) Double Pendulum; (c) Selkov; (d) Oscillator; (e) Hopf; (f) Lorenz96 & VL20.

where $\boldsymbol{Q} \in \mathbb{R}^{M\times M}$ and $\boldsymbol{R} \in \mathbb{R}^{N\times N}$ are the covariances. A summary of the essential characteristics of all the systems is provided in Table 4 and Figure 8. Following this, the specific background and mathematical modeling for each system are discussed in detail.

**Tracking.** The Tracking system, also known as the Wiener velocity model, is a canonical linear Gaussian system [5] commonly used for target Tracking [63]. The state represents the position and velocity of a moving object in two dimensions, expressed as $x = [p_x\ p_y\ v_x\ v_y]^\top$, where $p_x$ and $p_y$ denote the object's positions in the longitudinal and lateral directions, respectively, and $v_x$ and $v_y$ are the corresponding velocities. The observations are direct, noisy observations of the position components. The overall system can be described as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x},$$
$$\boldsymbol{y}_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{x}_t + \boldsymbol{\zeta}_t.$$

**Selkov.** The Selkov system models the dynamics of the glycolysis process through a set of two coupled differential equations [49]. The state vector $\boldsymbol{x} = [\theta_1\ \theta_2]^\top$ represents the concentrations of the metabolites involved in the glycolytic cycle, where $\theta_1$ and $\theta_2$ correspond to the concentrations of the key metabolites. The constants $a = 0.08$ and $b = 0.6$ are parameters related to the reaction rates within the system. The dynamics of the system and the observation process are governed by the following equations:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -\theta_1 + a\theta_2 + \theta_1^2\theta_2 \\ b - a\theta_2 - \theta_1^2\theta_2 \end{bmatrix}, \quad \boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{\zeta}_t.$$

**Damped Oscillator.** The damped Oscillator refers to a physical system in which an object experiences Oscillatory motion that gradually decreases over time due to the presence of a damping force [50]. In this system, the state vector $\boldsymbol{x} = [\theta_1\ \theta_2]^\top$ represents the positions of the Oscillator along two axes, $\theta_1$ and $\theta_2$. The overall system can be described as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -a\theta_1^3 + b\theta_2^3 \\ -b\theta_1^3 - a\theta_2^3 \end{bmatrix}, \quad \boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{\zeta}_t.$$

Here, $a = 0.1$ and $b = 2$ are constants that define the strength of the nonlinearity, and the cubic terms represent the restoring forces in the system.

**Hopf Bifurcation.** The Hopf bifurcation describes a dynamical system that undergoes spontaneous oscillations as a system parameter $\mu$ crosses a critical threshold, transitioning from a stable equilibrium to an Oscillatory regime [32]. This phenomenon is commonly observed in various real-world systems, such as chemical reactions, electrical circuits, and fluid dynamics. In this system, the state vector is represented as $\boldsymbol{x} = [\theta_1\ \theta_2]^\top$. The parameter $\mu = 0.5$ controls the stability of the system, while $\omega = 1$ and $A = 1$ are constants that define the frequency of oscillations and the strength of the nonlinear interactions, respectively. The overall system can be described as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \mu\theta_1 + \omega\theta_2 - A\theta_1(\theta_1^2 + \theta_2^2) \\ -\omega\theta_1 + \mu\theta_2 - A\theta_2(\theta_1^2 + \theta_2^2) \end{bmatrix}, \quad \boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{\zeta}_t.$$

**Double Pendulum.** The Double Pendulum, often referred to as the chaotic pendulum, is a nonlinear dynamical system consisting of a pendulum with another pendulum attached to its end [51]. The state vector $\boldsymbol{x} = [\theta_1\ \omega_1\ \theta_2\ \omega_2]^\top$ encapsulates the system's configuration, where $\theta_1$ and $\theta_2$ represent the angular positions, and $\omega_1$ and $\omega_2$ denote the angular velocities of the two pendulums. The system's observations are the noisy observations of the angles $\theta_1$ and $\theta_2$:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta_1} \\ \dot{\omega_1} \\ \dot{\theta_2} \\ \dot{\omega_2} \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \frac{M_2 L_1 \omega_1^2 \sin(D)\cos(D) + M_2 G \sin(\theta_2)\cos(D) + M_2 L_2 \omega_2^2 \sin(D) - (M_1+M_2) G \sin(\theta_1)}{(M_1+M_2) L_1 - M_2 L_1 \cos^2(D)} \\ \omega_2 \\ \frac{-M_2 L_2 \omega_2^2 \sin(D)\cos(D) + (M_1+M_2) G \sin(\theta_1)\cos(D) - (M_1+M_2) L_1 \omega_1^2 \sin(D) - (M_1+M_2) G \sin(\theta_2)}{\frac{L_2}{L_1}((M_1+M_2) L_1 - M_2 L_1 \cos^2(D))} \end{bmatrix},$$

$$\boldsymbol{y}_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}_t + \boldsymbol{\zeta}_t, \quad D = \theta_2 - \theta_1.$$

Here, $M_1 = 1$ and $M_2 = 1$ are the masses of the two pendulums, $L_1 = 1$ and $L_2 = 1$ represent their respective lengths, and $G = 9.8$ is the gravitational constant that governs the motion of the system.

**Lorenz96.** The Lorenz96 system, conceived by Edward Lorenz [52], offers a simplified yet powerful model of atmospheric dynamics, often serving as a cornerstone in chaos theory and meteorology. This system is described by a set of ordinary differential equations that capture the intricate interplay between atmospheric modes in a periodic domain:

$$\dot{\boldsymbol{x}}^{(j)} = (\boldsymbol{x}^{(j+1)} - \boldsymbol{x}^{(j-2)})\boldsymbol{x}^{(j-1)} - \boldsymbol{x}^{(j)} + F, \quad \boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{\zeta}_t, \quad j = 1, 2, \ldots, M.$$

In this model, $\boldsymbol{x}^{(j)}$ represents the $j$th state of the Lorenz96 system, while $F = 8$ is the external forcing term that drives the system. The system operates under periodic boundary conditions, ensuring continuity across the domain: $\boldsymbol{x}^{(M+1)} = \boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(M)}$, and $\boldsymbol{x}^{(-1)} = \boldsymbol{x}^{(M-1)}$, with $M = 72$ representing the total number of dimensions in the state space.

**Vissio-Lucarini 20.** The Vissio-Lucarini 20 model (VL20), introduced by Vissio and Lucarini [53], presents a coupled system that provides a minimalist yet rich representation of Earth's atmospheric dynamics. By extending the Lorenz framework to include *temperature* variables, the VL20 model enables the development of more complex and nuanced behavioral patterns:

$$\begin{aligned} \dot{\boldsymbol{\phi}}^{(j)} &= (\boldsymbol{\phi}^{(j+1)} - \boldsymbol{\phi}^{(j-2)})\boldsymbol{\phi}^{(j-1)} - \gamma\boldsymbol{\phi}^{(j)} - \varepsilon\boldsymbol{\theta}^{(j)} + F, \\ \dot{\boldsymbol{\theta}}^{(j)} &= \boldsymbol{\phi}^{(j+1)}\boldsymbol{\theta}^{(j+2)} - \boldsymbol{\phi}^{(j-1)}\boldsymbol{\theta}^{(j-2)} + \varepsilon\boldsymbol{\phi}^{(j)} - \gamma\boldsymbol{\theta}^{(j)} + G, \\ \boldsymbol{y}_t &= \boldsymbol{x}_t + \boldsymbol{\zeta}_t, \quad j = 1, 2, \ldots, M'. \end{aligned}$$

In this model, $M'$ represents the number of discrete points in the system, and the state vector is given by $\boldsymbol{x} = [\boldsymbol{\phi}\ \boldsymbol{\theta}]^\top$, with $M = 2M'$ variates. The constants $F = 10, G = 0, \gamma = 1, \varepsilon = 1$ are defined in [53]. Furthermore, the system is subject to periodic boundary conditions, akin to those in the Lorenz96 model.

## C Principle Analysis

In this section, we analyze the underlying connection between large language models (LLMs) and filtering. As illustrated in Figure 9, we present a simplified comparison of the two processes. LLMs predict the probability distribution of the next token based on all preceding tokens, while filtering estimates the distribution of the current system state given all past observations. This fundamental similarity, where both approaches make predictions based on sequential historical information, enables a natural alignment between them. With appropriate alignment, LLM-Filter can effectively leverage the reasoning capabilities and rich pre-trained knowledge of LLMs for state estimation tasks.

Regarding a more detailed theoretical explanation, it is challenging to provide one at this stage. This limitation stems from the fact that the theoretical foundations of LLMs themselves remain an open research problem [29, 64]. Consequently, many applications of LLMs across diverse domains—such as time series forecasting [30, 44, 46] and control systems [18, 19, 20, 65], also lack rigorous mathematical justification. Similarly, our work focuses on empirical evidence and the practical alignment between language modeling and filtering, while acknowledging the need for deeper theoretical understanding in future research.

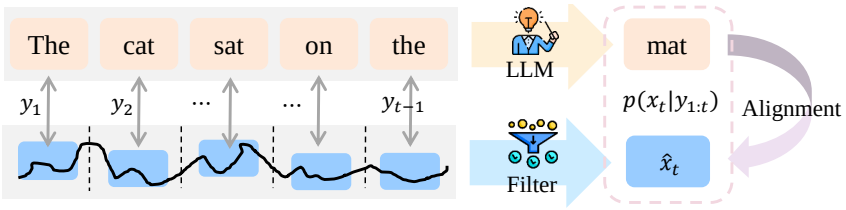



Figure 9: Principle Comparison of LLMs and Filtering. The LLM predicts the next token (e.g., *mat*) based on preceding tokens, while a filter estimates the current state $\hat{x}_t$ using past observations.

# D Experiment Details

Our model is built with PyTorch [66] and trained extensively on the same server with the following hardware configuration: 4 × Intel® Xeon® Gold 6226R @ 2.90GHz CPUs and 1 × NVIDIA H800-80G GPU. The complete code is publicly accessible at https://github.com/Anonymous-User-2367/LLM-Filter.

## D.1 Benchmark Settings

The online Bayes baseline methods are implemented by JAX [67] following [48]. The number of particles for EnKF, EnKFI, EnKFS, and HubEnKF is set to 1000 to balance efficiency and accuracy [47]. For a fair comparison, the hyperparameters of all online Bayes methods are selected on the first trial using the Bayesian Optimization (BO) package [55] with a parameter range of $(1, 20)$. The KalmanNet [13] implementation is based on their official open-source repository. The MEstimator [14], RStateNet [15], and ProTran [43] are implemented according to the methodologies outlined in their respective papers. The parameter counts and peak memory for different models in Table 5. Since we froze the core layers of Llama-7B and only trained the ObsEmbedding and StateProjection layers, we distinguish between the total parameter count and the trainable parameter count.

Our experiments are conducted on the systems described in Appendix B. We gather 100 trajectories, each consisting of 200 time steps, from each system, as shown in Table 4. To prevent data leakage, the datasets are split chronologically into training, validation, and test sets in a $7:1:2$ ratio.

Table 5: Comparison of Model Parameters and Peak Memory

| Model | LLM-Filter (Total/Trainable) | MEstimator | RStateNet | ProTran | KalmanNet |
|---|---|---|---|---|---|
| **Parameter** | 6.61B / 4.22M | 2.12M | 7.38M | 15.77M | 5.20K |
| **Peak memory** | 27.43GiB | 622.51MiB | 742.57MiB | 854.54MiB | 582.93MiB |

## D.2 Implementation

In LLM-Filter, only the $\mathrm{ObsEmbedding}$and $\mathrm{StateProjection}$ layers are updated via the gradient. These components are implemented using a simple MLP, which has been shown to effectively embed and project time series for LLM [30]. The default hidden dimension and number of MLP layers, batch size, window length $T$, and segment length $L$ are set to 512, 2, 16 ,40, and 20 respectively. We use the AdamW optimizer [68] with the initial learning rate of $10^{-4}$ and the weight decay of $10^{-5}$, where the model is trained for 10 epochs. Unless otherwise specified, we use LLaMA-7B [56] as the default base LLM.

## D.3 Evaluation Metrics

For evaluation, we use the root mean square error (RMSE) as the primary metric to assess the accuracy of state estimation [5]. To evaluate efficiency, we measure the inference running time per step in

milliseconds (ms/step). The calculations for these metrics are as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{T_{traj}} \|x_t - \hat{x}_t\|^2}{T_{traj}}}, \quad \text{Runtime} = \frac{\text{Cost Time}}{T_{traj} \cdot N_{test}},$$

where $T_{traj}$ is the trajectory length and $N_{test}$ denotes the number of trajectories in the test set.

# E Supplementary Results

## E.1 Running Time

We evaluated the estimation efficiency (ms/step) of various methods on the same device, as described in Appendix D. The results are presented in Table 6.

Table 6: Comparison of Estimation Runtime (ms/step) Across Methods.

| Method | LLM-Filter | EnKF | RStateNet |
|---|---|---|---|
| **Selkov** | 0.8766 | 2.5095 | 0.1546 |
| **Oscillator** | 0.8299 | 2.6815 | 0.1553 |
| **Hopf** | 0.8245 | 2.5747 | 0.1267 |
| **Pendulum** | 1.2447 | 2.2246 | 0.0967 |
| **Lorenz96** | 0.9577 | 6.7377 | 0.4513 |
| **VL20** | 0.9503 | 2.9397 | 0.4521 |

## E.2 Generalization Estimation Experiments

To assess the generalization capabilities of LLM-Filter, we conducted extensive evaluation. First, we evaluated its performance on systems subjected to varying strengths of outliers, as detailed in Table 7. Notably, while the performance of other methods deteriorated rapidly with an increasing number of outliers, LLM-Filter demonstrated minimal performance loss.

We also tested the classical observation-disturbance scenario [48], where robust filtering methods such as EnKFI, EnKFS, and HubEnKF are known to excel. As shown in Table 8, although LLM-Filter did not achieve the best results, it delivered performance comparable to specialized robust algorithms. These findings highlight LLM-Filter's ability to generalize effectively even in the presence of significant outlier interference or system turbulence, offering a reliable solution across diverse and challenging scenarios.

In addition, we provide the complete experimental results for the cross-system scenarios discussed in Section 5.2, as shown in Table 9.

## E.3 Scaling Behavior

We investigate the scaling behavior of LLM-Filter in state estimation tasks by evaluating models with various pretrained LLM backbones. Detailed results can be found in Table 10. Our analysis reveals that LLM-Filter follows scaling-law behavior: as the parameter size of the model increases, the average estimation error tends to decrease, leading to improved estimation performance.

## E.4 Hyperparameter Sensitivity

In the main experiments, the configuration of the LLM-Filter is provided in Appendix D.2. Here, we verify the robustness of the LLM-Filter with respect to hyperparameters, including the sliding length $T$, the hidden layer dimension, and the layer number of MLP in $\mathrm{ObsEmbedding}$ and $\mathrm{StateProjection}$. Experiments were conducted on the Selkov and VL20 systems, with the results summarized in Figure 10. Overall, LLM-Filter demonstrates consistent performance across different parameter settings, showing it is largely insensitive to hyperparameter variations. On average, a window length of 20 or 40 yields better results. This suggests that appropriately chosen window lengths strike a balance—shorter windows risk losing crucial information, while longer windows may fail to capture local features effectively.

Table 7: RMSE Results for Observation Outlier Scenarios. Here $R$ denotes the true observation covariance with outliers, while $R_0$ represents the assumed covariance.

| Method | $R$ | LLM-Filter | EnKF | EnKFI | EnKFS | HubEnKF |
|---|---|---|---|---|---|---|
| **Selkov** | $R_0$ | **0.4061** | 0.5978 | 0.6703 | 0.6081 | 0.6676 |
| | $25R_0$ | **0.6898** | 1.8626 | 1.9874 | 1.6523 | 2.1998 |
| | $50R_0$ | **0.7034** | 2.5496 | 3.4248 | 2.6597 | 2.5318 |
| | $75R_0$ | **0.7074** | 3.0952 | 4.3829 | 3.1699 | 2.8636 |
| | $100R_0$ | **0.7090** | 3.5625 | 4.8417 | 4.3972 | 2.9850 |
| **Oscillator** | $R_0$ | **0.5247** | 0.5505 | 0.5516 | 0.5548 | 0.5540 |
| | $25R_0$ | **0.5946** | 1.1630 | 1.2714 | 1.1674 | 1.1746 |
| | $50R_0$ | **0.6044** | 1.3267 | 1.3195 | 1.3575 | 1.4778 |
| | $75R_0$ | **0.6083** | 1.4701 | 3.1428 | 1.4757 | 1.8707 |
| | $100R_0$ | **0.6110** | 1.6154 | 1.6040 | 1.5793 | 2.1680 |
| **Hopf** | $R_0$ | **0.5751** | 0.6322 | 0.6983 | 0.6380 | 0.6982 |
| | $25R_0$ | **0.7142** | 0.9946 | 1.1299 | 0.9938 | 1.0976 |
| | $50R_0$ | **0.7738** | 1.0549 | 1.1657 | 1.2035 | 1.1155 |
| | $75R_0$ | **0.8010** | 1.0898 | 1.1872 | 1.5071 | 1.1313 |
| | $100R_0$ | **0.8096** | 1.1146 | 1.1989 | 2.1321 | 1.1483 |

Table 8: RMSE Results of Observation-Disturbance Scenarios.

| Method | LLM-Filter | EnKF | EnKFI | EnKFS | HubEnKF |
|---|---|---|---|---|---|
| **Selkov** | **0.4999** | 0.9223 | 0.9500 | 0.6079 | 0.6198 |
| **Oscillator** | 0.7451 | 1.3431 | 1.3611 | 0.5543 | **0.5364** |
| **Hopf** | 0.6827 | 0.8220 | NaN | 0.6379 | 0.7226 |
| **Lorenz96** | **0.9186** | 7.6225 | 7.7459 | 4.5833 | 3.1399 |
| **VL20** | **0.8876** | NaN | 7.7777 | 11.5041 | 0.9684 |

Table 9: RMSE of Cross-System Scenarios.

| Method | LLM-Filter | LLM-Filter-O | MEstimator | RStateNet | ProTran |
|---|---|---|---|---|---|
| **Tracking→Pendulum** | **1.2568** | 1.6573 | 3.6957 | 3.1052 | 3.1162 |
| **Pendulum→Tracking** | **1.4214** | 1.8936 | 2.7407 | 5.0927 | NaN |
| **Oscillator→Hopf** | **0.6613** | 2.1125 | NaN | 3.8645 | 3.5668 |
| **Hopf→Oscillator** | **0.9069** | 1.5624 | 2.5369 | 2.7542 | 3.0075 |

Table 10: RMSE Results of Various LLMs in the LLM-Filter Framework.

| LLM | GPT-2 (124M) | OPT-1.3B | LLaMA-7B | LLaMA-13B |
|---|---|---|---|---|
| **Selkov** | 1.0434 | 1.1713 | 0.6369 | **0.6200** |
| **Oscillator** | 0.9393 | 1.0702 | 0.5753 | **0.4441** |
| **Hopf** | 0.9611 | 0.8466 | 0.8180 | **0.7000** |
| **Lorenz96** | 0.9727 | 0.9029 | 0.9735 | **0.9147** |
| **VL20** | 0.9931 | **0.5999** | 0.8433 | 0.7623 |
| **Average** | 0.9819 | 0.9182 | 0.7694 | **0.6882** |

Additionally, the segment length in LLM-Filter is a hyperparameter that determines how many observations are grouped into a single token before being fed into the LLM. We conducted further experiments using different segment lengths while keeping the window length fixed at 40, as shown in Table 11. The performance of LLM-Filter remains relatively stable across different segment lengths, demonstrating its robustness to this hyperparameter.

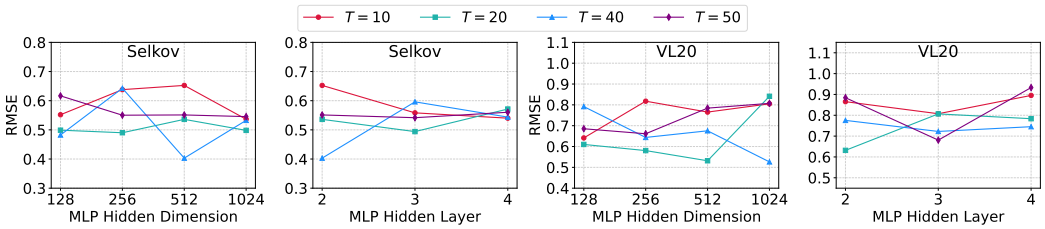



Figure 10: Hyperparameter Sensitivity of LLM-Filter on the Selkov and VL20 Systems.

Table 11: RMSE Comparison with Varying Segment Lengths

| **Segment Length** | **Base (20)** | **10** | **5** |
|---|---|---|---|
| Selkov | 0.4061 | 0.7024 | 0.3077 |
| Oscillator | 0.5247 | 0.5829 | 0.5298 |
| Hopf | 0.5751 | 0.6222 | 0.5779 |
| Pendulum | 0.8348 | 0.5028 | 0.8254 |
| Lorenz96 | 0.9149 | 0.9200 | 0.9182 |
| VL20 | 0.7717 | 1.0346 | 1.0221 |
| **Average** | **0.6712** | **0.7275** | **0.6969** |