Trends and Tests on Hybrid Physics-Informed Machine Learning Techniques

Dumont Jules

August 9, 2025

Abstract

This report explores hybrid Physics-Informed Machine Learning (PIML) techniques, focusing on their application to chaotic dynamical systems. We investigate the integration of physical laws into machine learning models to enhance prediction accuracy and state reconstruction, particularly for unmeasured variables. The primary aim is to analyze the Physics-Informed Long Short-Term Memory (PI-LSTM) network, as detailed in the paper "Physics-Informed Long Short-Term Memory for Forecasting and Reconstruction of Chaos" by Özalp et al. (2023). We detail its architecture, the formulation of its physics-informed loss, and its performance in reconstructing unmeasured variables and Lyapunov exponents of the Lorenz-96 model. Furthermore, we propose a conceptual framework for a Transformer-based hybrid model, discussing its potential advantages and architectural considerations as an alternative to recurrent neural networks for such tasks. The report emphasizes the benefits of PIML in achieving higher-precision forecasts and physically-interpretable surrogates, especially in data-scarce or partially observed scenarios.

Contents

Chapter 1

Introduction and Hybrid PIML Model Categories

1.1 Context and Motivation

In many scientific and engineering disciplines, particularly in meteorology, oceanography, and climate re-analysis, understanding and predicting the evolution of complex dynamical systems is crucial. Often, only partial information about the system's state is available due to computational costs or sensor limitations, necessitating the reconstruction of unmeasured or hidden variables. Traditional data-driven machine learning models, while powerful, can struggle with generalization and physical consistency, especially in chaotic systems where exponential sensitivity to initial conditions makes long-term accurate prediction difficult.

Hybrid Physics-Informed Machine Learning (PIML) techniques address these challenges by integrating knowledge of governing physical equations directly into the learning process. This approach leads to higher-precision forecasts and physically-interpretable models. The physics-based loss term acts as a regularization, penalizing solutions that violate the system's underlying differential equations. This can be viewed as a "parallelized" numerical integration of the governing differential equations, guiding the model towards physically consistent solutions. By embedding prior physical knowledge, hybrid models reduce the reliance on extensive data, maintain physical fidelity, and combine the strengths of both data-driven and physics-based modeling paradigms. This is particularly advantageous in scenarios with limited observational data or when aiming for robust long-term predictions in chaotic environments.

1.2 Scope of the Report

This report focuses on the application of hybrid machine learning techniques to chaotic dynamical systems, with a particular emphasis on state reconstruction and long-term forecasting. We will primarily analyze the Physics-Informed Long Short-Term Memory (PI-LSTM) network, drawing insights from the paper "Physics-Informed Long Short-Term Memory for Forecasting and Reconstruction of Chaos" by Özalp et al. (2023). The report will detail the PI-LSTM architecture, the formulation of its physics-informed loss, and its performance in reconstructing unmeasured variables and Lyapunov exponents of the Lorenz-96 model. Additionally, we will explore the potential of Transformer-based architectures for similar physics-informed tasks, discussing their theoretical advantages over recurrent networks and outlining a conceptual framework for their implementation in a hybrid PIML context. The objective is to provide a concise, factual overview of these techniques.

1.3 Introduction to AutoDiff

Forward-mode automatic differentiation (AD) is easiest to picture with the idea of *dual numbers*. You take your normal input x and add a tiny "ghost" ε that satisfies $\varepsilon^2 = 0$. Then you run the *same* code once at $x + \varepsilon$ and—almost magically—the coefficient of ε carries the exact derivative. No limits, no symbolic algebra, and no need to sample a second point.

Finite difference vs. AD. In a classic finite-difference mindset you approximate

$$f'(x) \;\approx\; \frac{f(x+h) - f(x)}{h}$$

using two close points x and $x + h$. Forward-mode AD threads the dual number *inside* every operation, so it sweeps the whole program once and gives the derivative "for free" alongside the original value.

Tiny example. Let $f(x) = x^2$. Evaluate at $x + \varepsilon$:

$$f(x+\varepsilon) = (x+\varepsilon)^2 = x^2 + 2x\,\varepsilon,$$

and because $\varepsilon^2 = 0$ the $2x$ in front of ε is exactly $f'(x)$. The same trick works for any polynomial—and since a neural network is basically a giant nested polynomial (affine layers plus element-wise nonlinearities), AD scales naturally. Modern libraries hide the ε bookkeeping; you just tag your input and let the engine propagate values and derivatives together.

1.4 Data-Driven Parameter Inference for Differential Equations

This type of Machine Learning technique is often refered as PINN focus on identifying the coefficients of a differential equation from data, allowing next to use the learned coefficents to then make a standard numerical integration of the differential equation in order to make predicitons. A bite-sized illustration is a physics-informed neural network (PINN) for the damped spring–mass oscillator [3]. Let a tiny MLP $x_\theta(t)$ stand in for the displacement. Autodiff gives the residual

$$r_\theta(t) = m\ddot{x}_\theta + c\dot{x}_\theta + kx_\theta,$$

and the loss just balances data fit with physics

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\big[x_\theta(t_i) - x_i^{\text{obs}}\big]^2 + \lambda\frac{1}{N}\sum_{i=1}^{N} r_\theta(t_i)^2.$$

Gradient descent nudges both the weights θ *and* the unknown coefficients (m, c, k) in one go, so once trained you can drop the net into an RK4 solver and roll the system forward.

1.5 Physics-Constrained Neural Fields via Autodiff

Physics-Constrained Neural Fields, such as DeepPhysNet [2], constitute a branch of PIML in which a neural field (an MLP that maps continuous space-time coordinates to physical states) is embedded in a larger, physics-aware architecture. In DeepPhysNet a Transformer-based hypernetwork ingests coarse-resolution meteorological sequences (plus static geographic features) and outputs part of the weights of several per-variable physics networks. Each physics network then accepts coordinates (x, y, t) and returns the corresponding surface variables (u, v, T, p, ρ, q)

Training combines (i) a regression term on grid- or station-level observations and (ii) a soft-constraint term that evaluates simplified momentum, continuity, thermodynamic, water-vapour and ideal-gas equations through automatic differentiation, ensuring that the learned field respects atmospheric physics at every point in the domain. Because time is treated explicitly

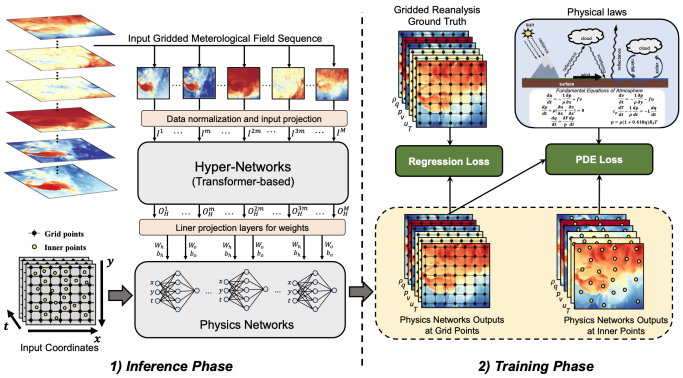


Figure 1.1: DeepPhysNet Framework

as a coordinate and the hyper-network provides rich temporal context, DeepPhysNet supports continuous downscaling, bias-correction and short-range forecasting, thereby overcoming the sequential-dependency and resolution-scaling limitations that plain fully-connected neural fields face in chaotic dynamical systems.

1.6 Sequential Physics-Guided State Reconstruction Models

Sequential physics-guided models, such as the Physics-Informed Long Short-Term Memory (PI-LSTM) network [4], are designed to handle time-dependent data and reconstruct unmeasured variables in dynamical systems. These models leverage recurrent architectures (like LSTMs) to capture temporal dependencies while incorporating physical constraints. A key aspect is their ability to infer unobserved variables even when these variables are not explicitly present in the training targets. The physical laws guide the network to learn the complete system dynamics, allowing for the reconstruction of the full state from partial observations. This is the model we will focus on and implement.

Chapter 2

Implementation of LSTM and Transformer Hybrid Models

2.1 LSTM-Based Hybrid Model

2.1.1 LSTM Architecture and Training Basics

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to handle sequential data and overcome the vanishing/exploding gradient problems inherent in traditional RNNs. Unlike fully-connected networks, LSTMs share weights across time steps, making them suitable for processing sequences of arbitrary length. An LSTM cell maintains a cell state (c_t) and a hidden state (h_t), which are updated at each step based on the current input and previous states. The core of an LSTM's ability to manage long-range dependencies lies in its gating mechanisms: the input gate, forget gate, and output gate. These gates, typically sigmoid functions, control the flow of information into and out of the cell state, allowing the network to selectively remember or forget information over long sequences. This mechanism, introduced by Hochreiter and Schmidhuber (1997) [1], effectively mitigates vanishing gradients. The hidden state (h_t) can be interpreted as a compressed memory of past dynamics, enabling the network to learn complex temporal patterns. Training LSTMs involves Back-Propagation Through Time (BPTT), where gradients are computed by unrolling the network over time steps.

2.1.2 Open-Loop vs. Closed-Loop Modes

The operation of sequential models like LSTMs can be distinguished into two primary modes: open-loop (teacher forcing) and closed-loop (autonomous prediction).

- **Open-Loop (Teacher Forcing):** During training and validation, the network typically operates in an open-loop configuration. At each time step t_i, the true observed data $x(t_i)$ is fed as input to the LSTM cell. The network then predicts the state at t_{i+1}. This approach, known as teacher forcing, stabilizes training by providing the ground truth at each step, preventing the accumulation of prediction errors.
- **Closed-Loop (Autonomous Prediction):** After training, for long-term forecasting or evaluation on test data, the network operates in a closed-loop configuration. The process begins with an initial true input. For subsequent steps, the network's own predicted observed variables ($\tilde{x}(t_{i+1})$) are fed back as input to the LSTM cell. This allows for an autonomous evolution of the LSTM, effectively defining a dynamical system whose long-term behavior can be analyzed.

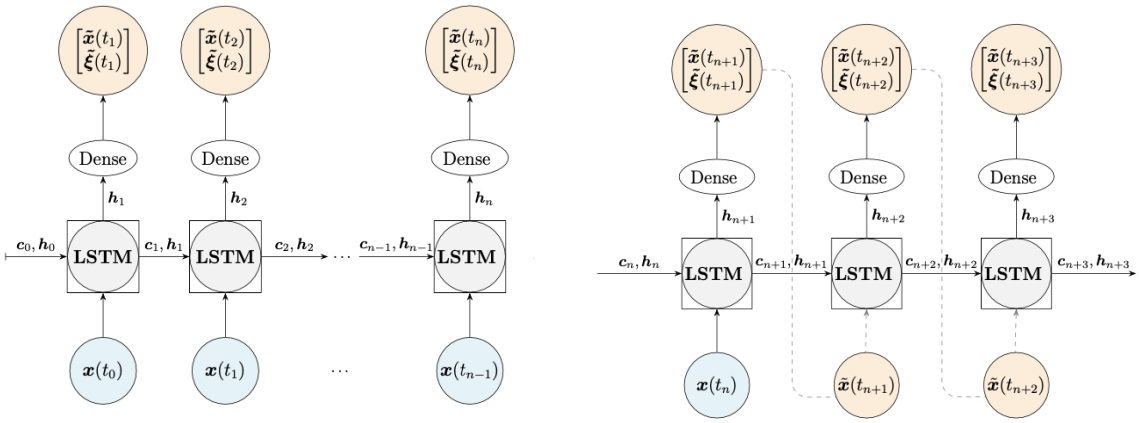


Figure 2.1: Open-loop (a) and closed-loop (b) configurations.

2.1.3 State Reconstruction Framework

For state reconstruction, the PI-LSTM aims to predict the full state $y(t_{i+1}) = [x(t_{i+1}); \xi(t_{i+1})]$ from partial observations $x(t_i)$, where $x(t)$ are the observed variables and $\xi(t)$ are the unmeasured variables. The LSTM's hidden state h_{i+1} encapsulates the learned temporal dependencies. This hidden state is then passed through a dense layer to obtain the full state prediction:

$$[\tilde{x}(t_{i+1}); \tilde{\xi}(t_{i+1})] = W^{\text{dense}} h_{i+1} + b^{\text{dense}}$$

Here, $W^{\text{dense}} \in \mathbb{R}^{(N_x+N_\xi)\times N_h}$ and $b^{\text{dense}} \in \mathbb{R}^{N_x+N_\xi}$, where N_x is the dimension of observed variables, N_ξ is the dimension of unmeasured variables, and N_h is the hidden state dimension. This framework allows the network to infer the unobserved variables $\tilde{\xi}(t_{i+1})$ at each time step, even though they are never directly provided as training targets.

2.1.4 Physics-Informed Loss for State Reconstruction

To constrain the network to respect the underlying physical laws, a physics-informed loss term is added to the data-driven loss. The total loss function L is a weighted sum of the data-driven loss (L_{dd}) and the physics-informed loss (L_{pi}):

$$L = L_{dd} + \alpha_{pi} L_{pi} \quad (2)$$

where $\alpha_{pi} \in \mathbb{R}^+$ is a penalty hyperparameter. If $\alpha_{pi} = 0$, the network operates as a purely data-driven LSTM.

The data-driven loss L_{dd} is typically a mean-squared error on the observed data:

$$L_{dd} = \frac{1}{N_t} \sum_{i=1}^{N_t} (x(t_i) - \tilde{x}(t_i))^2$$

The physics-informed loss L_{pi} penalizes violations of the system's governing equations, $d/dty(t) = f(y(t))$:

$$L_{pi} = \frac{1}{N_t} \sum_{i=1}^{N_t} \left(\frac{d}{dt}\tilde{y}(t_i) - f(\tilde{y}(t_i)) \right)^2$$

For simplicity, the time derivative $d/dt\tilde{y}$ is often computed using a forward difference scheme:

$$\frac{d}{dt}\tilde{y}(t_i) \approx \frac{\tilde{y}(t_{i+1}) - \tilde{y}(t_i)}{\Delta t}$$

where $\tilde{y}(t_i)$ is the network's prediction. This loss regularizes the network's training to provide predictions that fulfill the governing equations up to a numerical tolerance. A small α_{pi} implies that the network learns primarily from data first, with physics acting as a soft constraint, while a larger α_{pi} enforces stronger adherence to physical laws. For the Lorenz-96 model, variables are typically z-score-normalized before training, as described in Section 2.2.1.

2.1.5 Closed-Loop Inference Procedure

After training with fixed weights and biases, the PI-LSTM is evaluated in a closed-loop configuration for long-horizon forecasts. The process starts with an initial observed state $x(t_n)$. The network predicts the full state $[\tilde{x}(t_{n+1}); \tilde{\xi}(t_{n+1})]$. For the next time step, the predicted observed part $\tilde{x}(t_{n+1})$ is fed back as input to the LSTM. This autonomous feedback loop allows the network to generate extended trajectories without further ground truth data, enabling the study of its long-term ergodic properties and stability.

2.1.6 Lyapunov Exponent Analysis

Lyapunov Exponents (LEs) are crucial quantities for characterizing chaotic dynamical systems, quantifying the average rate of divergence or convergence of nearby trajectories in phase space. Chaotic systems possess at least one positive LE. By reconstructing the unmeasured variables, the PI-LSTM effectively reconstructs the tangent space of the system. Computing the LEs from the trained network's autonomous evolution allows for assessing how well the model reproduces the chaotic dynamics of the original system. Agreement between the network's LEs and the true system's LEs indicates that the PI-LSTM has learned the fundamental stability properties of the chaotic attractor.

Distribution Similarity: Jensen–Shannon Distance Table

To compare long-term statistical fidelity, we report the Jensen–Shannon distances (base-2) between the PDFs of model-generated trajectories and the ground truth, aggregated across simulations for each configuration. Lower values indicate closer agreement. The table below is auto-generated from the closed-loop metrics.

Table 2.1: Jensen–Shannon distances (base-2) across models in closed-loop mode. Values are mean ± std across simulations; lower is better.

Model	$\mathrm{JS}_{\mathrm{mean}}$	$\mathrm{JS}_{\mathrm{max}}$	Runs
LSTM_alphaPI0.01_seqLen200_epochs40	0.0090 ± 0.0006	0.0137 ± 0.0009	2
LSTM_alphaPI0.01_seqLen201_epochs100	0.0095 ± 0.0001	0.0136 ± 0.0001	2
LSTM_alphaPI0.01_seqLen50_epochs100	0.0312 ± 0.0178	0.0438 ± 0.0212	2
LSTM_alphaPI0.1_seqLen50_epochs100	0.1162 ± 0.0146	0.2976 ± 0.0028	2

Lyapunov Exponent Computation Algorithm (QR-based)

Let the system evolve as a discrete map:

$$\mathbf{y}_{n+1} = F(\mathbf{y}_n)$$

Let $J_n = \frac{\partial F}{\partial \mathbf{y}_n}$ be the Jacobian at step n.
We track the evolution of d independent perturbation vectors.

1. **Initial orthonormal basis:**

$$Q_0^{\mathrm{old}} = [\mathbf{q}_1^{(0)}, \ldots, \mathbf{q}_d^{(0)}], \quad \text{with } (Q_0^{\mathrm{old}})^\top Q_0^{\mathrm{old}} = I$$

2. **At each time step n:**

   (a) Propagate the basis vectors using the Jacobian:

$$W_n = J_n Q_n^{\text{old}}$$

   (b) Apply QR decomposition to re-orthonormalise:

$$W_n = Q_{n+1}^{\text{new}} R_n$$

   where:

   - Q_{n+1}^{new} is an orthonormal matrix
   - R_n is upper-triangular with positive diagonals

   (c) Accumulate logarithmic stretch factors:

$$s_k \leftarrow s_k + \ln R_n[k,k], \quad \text{for } k = 1, \ldots, d$$

3. **After T steps, compute the Lyapunov exponents:**

$$\lambda_k = \frac{1}{T \cdot \Delta t} \sum_{n=0}^{T-1} \ln R_n[k,k] \quad \text{for } k = 1, \ldots, d$$

For a machine learning model like the PI-LSTM, the Jacobian J_n can be computed using automatic differentiation.

2.1.7 Probabilistic Parameter Diagnostics

To verify the dynamic fidelity of the reconstructed variables, probabilistic parameter diagnostics are employed, primarily through the comparison of Probability Density Functions (PDFs). By comparing the PDFs of the reconstructed unmeasured variables (e.g., $\tilde{\xi}(t)$) from the PI-LSTM's autonomous trajectory with the target PDFs from the true system, one can assess the model's ability to capture the correct long-term statistical behavior. Deviations in PDFs, such as the collapse to a fixed point (delta-like distribution) observed in purely data-driven LSTMs, indicate a qualitative failure in reproducing the system's dynamics. Quantitative metrics like Kullback-Leibler (KL) divergence can be used to measure the difference between the distributions. Q-Q plots can also be used to assess agreement, particularly for heavy-tail distributions.

2.2 Model Implementation

2.2.1 Data Normalization Strategies

Data normalization is critical for stable and efficient training of neural networks. For time series data in dynamical systems, a standard z-score normalization is commonly applied. This involves transforming the data x to have zero mean and unit variance:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation, respectively, computed exclusively from the training set. These statistics (μ, σ) are then fixed and used for normalizing both validation and test data during inference, ensuring consistency.

2.2.2 Denormalization of Predictions for Loss Computation

An important aspect of especially with physics-informed models is to ensure that the loss is computing on the original scale of the data. After the model makes predictions, the predicted values are denormalized back to their original scale before computing the loss. This step isn't ussually required for the data loss in standard model. But here with the introduction of the physics-informed loss using a differential equation, especially in presence of non-linearities, using normalized values will lead to incorrect gradients and thus incorrect training. Another strategies could be to adapt the differential equation to normalized values, but this won't be done here.

2.2.3 Training Schedule and Unroll Horizon

The training of LSTMs for time series involves processing sequences. Instead of feeding the entire sequence at once, which can be computationally expensive and memory-intensive, sequences are typically chunked into mini-batches. Each mini-batch consists of shorter sequences, and the network is trained using Truncated Back-Propagation Through Time (TBPTT). TBPTT involves unrolling the LSTM for a fixed number of steps (e.g., 50-100 steps, as commonly seen in literature) before computing gradients and performing a weight update. This trade-off balances GPU memory constraints with the need for sufficient temporal context to learn long-range dependencies. The choice of unroll horizon impacts the stability of training and the network's ability to capture long-term dynamics.

2.2.4 Influence of Input Sequence Length

The length of the input sequence (observed window) provided to the LSTM at each time step can significantly affect the state reconstruction quality. A longer input sequence provides more historical context, potentially enabling the network to better infer the underlying dynamics and reconstruct unmeasured variables. However, excessively long sequences can increase computational cost and might not always lead to proportional gains in accuracy, especially if the system's memory is shorter than the sequence length. Experimenting with varying observed window lengths is crucial to find an optimal balance between information content and computational efficiency for a given chaotic system.

2.3 Transformer-Based Hybrid Model

2.3.1 Transformers as Successors to RNNs and LSTMs

Transformers have emerged as powerful alternatives to Recurrent Neural Networks (RNNs) and LSTMs, particularly in tasks requiring long-range dependency modeling. Unlike RNNs that process sequences sequentially, Transformers rely entirely on self-attention mechanisms, allowing them to process all elements in a sequence in parallel. This parallelism significantly speeds up training and inference on modern hardware. The self-attention mechanism enables each element in the sequence to attend to all other elements, effectively capturing global dependencies regardless of their distance in the sequence. This excels at handling long-range dependencies, a common challenge for LSTMs. However, standard Transformers lack an explicit recurrent mechanism or memory of past states beyond the current input sequence, which can be a limitation for very long time series. Variants like Transformer-XL or MemTransformer address this by incorporating recurrence or memory mechanisms. A drawback of the original self-attention mechanism is its quadratic computational cost with respect to sequence length, $O(L^2)$, which can be prohibitive for very high-resolution domains. Efficient Transformer variants, such as Performer or Linformer, aim to reduce this cost to linear or near-linear complexity.

2.3.2 Architecture: Encoder–Decoder

For time series forecasting and state reconstruction, an encoder-decoder Transformer architecture is chosen due to its ability to effectively mimic the sequential processing and output generation of LSTM models. This architecture comprises two main components: an encoder and a decoder.

- **Encoder:** The encoder processes the input sequence $X_{\text{in}} = [x(t_0), x(t_1), \ldots, x(t_{L-1})]$, where L is the input sequence length, analogous to the number of inputs of the LSTM. The encoder operates in parallel across all time steps in the input sequence. Each element in the input sequence is processed simultaneously through multiple self-attention layers, allowing the model to capture global dependencies and contextual information across the entire input window in a single pass. This parallel processing is a key advantage over recurrent networks, which process inputs step-by-step. The output of the encoder is a rich, context-aware representation of the input sequence. The encoder is producing a latent sequence $Z_(1 : T)$, and is given a new input after the decoder finished its rollout and the backpropagation is done.
- **Decoder:** The decoder is trained to predict the full state $Y_{\text{target}} = [\tilde{y}(t_0), \tilde{y}(t_1), \ldots, \tilde{y}(t_L)]$. The decoder generates its output in parralel, it is trained to both reconstruction and next step prediciton. This will allow straightforward closed-loop inference. In closed-loop mode, the predicted output for the observed variables at time t_k can be fed back into the model as input for predicting the state at t_{k+1}, enabling long-horizon forecasts. The decoder also utilizes cross-attention mechanisms to attend to the encoder's output, integrating the learned context from the input sequence. The decoder is also using a causal mask, ensuring that predictions at time t_k do not depend on future information.

This architecture is well-suited for sequence-to-sequence tasks, where the encoder processes the observed history and the decoder generates the future states, including the unmeasured variables, while maintaining the ability for autonomous long-term prediction. (This let us with a latent sequence encoder and et autoregressiv decoder with causal mask)

Architecture feed-forward à reconstruction-prédiction Le modèle prend en entrée une séquence d'états partiellement observés

$$x(1:T) \in \mathbb{R}^{T \times d_{\text{obs}}}$$

et génère en sortie une séquence complète d'états reconstruits et prédits :

$$\tilde{y}(1:T+1) \in \mathbb{R}^{(T+1) \times d_{\text{full}}}$$

Le processus est le même pendant l'entraînement et l'inférence :

$$\begin{aligned} z(1:T) &= \text{Encoder}\big(x(1:T)\big) \\ \tilde{y}(1:T+1) &= \text{Decoder}(z(1:T)) \end{aligned}$$

où :

- Encoder extrait une représentation latente z (séquentielle) à partir de la séquence partielle,
- Decoder reconstruit l'état complet à chaque pas de temps et prédit l'état suivant à l'aide d'un token future,
- aucune sortie $\tilde{y}_t$ n'est utilisée en entrée du modèle, ni pendant l'entraînement, ni pendant l'inférence,
- le modèle est entièrement *feed-forward*, sans mécanisme autoregressif, et donc sans masque causal.

2.3.3 Positional Encoding Mechanisms

Since Transformers process sequences in parallel without inherent recurrence, they lose information about the order of elements. Positional encodings are added to the input embeddings to reintroduce this crucial sequential information.

- **Sinusoidal Positional Encodings:** These are fixed functions (sine and cosine waves of different frequencies) used to generate unique positional vectors that are added to the input embeddings. The i-th component of the positional encoding for position p is given by:

$$PE(p, 2i) = \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right)$$

$$PE(p, 2i+1) = \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right)$$

  where p is the position and d_{model} is the dimension of the model. These encodings are non-learnable and have the advantage of generalizing well to sequence lengths longer than those seen during training, as they are based on a mathematical function rather than learned embeddings.

2.3.4 Training Schedule Adapted for Transformers

Training Transformers for time series forecasting and state reconstruction requires adapting the training schedule from typical NLP tasks, with particular attention to memory management.

- **Sequence Length and Batch Size:** As previously mentioned in Section 3.1, the self-attention mechanism's quadratic computational cost with respect to sequence length ($O(L^2)$) means that longer sequences can quickly lead to significant memory problems, especially for high-resolution or long-horizon forecasting tasks. The choice of sequence length and batch size is a critical trade-off between capturing long-range dependencies and GPU memory constraints. Careful selection of sequence length is necessary to balance the need for sufficient temporal context with available hardware resources.
- **Gradient Clipping:** To ensure training stability, especially with deep networks and potentially large gradients from physics-informed losses, gradient clipping is often employed. This helps prevent exploding gradients that can arise during training.
- **Masking:** For generative tasks or when predicting future states, appropriate masking mechanisms (e.g., causal masking in decoder-only models) are essential to prevent the model from "cheating" by looking at future information.

2.4 Experimental Setup

2.4.1 Lorenz–96 Model Configuration

The Lorenz-96 model is a system of coupled ordinary differential equations that serves as a prototypical chaotic dynamical system, often used to model atmospheric dynamics. Its formulation is given by:

$$\frac{d}{dt}y_i(t) = (y_{i+1}(t) - y_{i-2}(t))y_{i-1}(t) - y_i(t) + F, \quad i = 1, \ldots, N \quad (3)$$

with periodic boundary conditions: $y_{-1}(t) = y_{N-1}(t)$, $y_0(t) = y_N(t)$, and $y_{N+1}(t) = y_1(t)$. For this study, the configuration parameters are set as:

- Dimension $N = 10$

- External forcing $F = 8$
- Integration time step $\Delta t = 0.01$ (for numerical solution and reference LEs)

This configuration results in a chaotic system with three positive Lyapunov exponents, the largest of which is approximately $\lambda_1 \approx 1.59$. The training set consists of $N_t = 20000$ points, equivalent to approximately $125\tau_\lambda$ (where $\tau_\lambda = 1/\lambda_1$ is the Lyapunov time). The data is split into training and validation windows, with a separate test set for closed-loop evaluation. Future work could explore harder regimes, such as $N = 40$, which exhibit more complex chaotic behavior.

2.4.2 LSTM Model Size and Hyperparameters

The Physics-Informed LSTM (PI-LSTM) model used for state reconstruction is configured with the following hyperparameters, optimized through tuning:

- Hidden and cell state dimension $(N_h) = 100$
- Sequence length = 200
- Physics-informed loss penalty hyperparameter $(\alpha_{pi}) = 0.01$

The total number of parameters of the LSTM is ?? The model will be trained on 100 epochs with a batch size of 64.

2.4.3 Transformer Model Size and Hyperparameters

For the proposed Transformer-based hybrid model, we will explore two configurations one fair in comparison to the LSTM:

- Number of encoder/decoder layers = 2
- Number of attention heads = 4
- Model dimension $(d_{\text{model}}) = 28$
- Feed-forward network dimension $(d_{\text{ff}}) = 112$
- Dropout rate = 0.1
- Physics-informed loss penalty hyperparameter $(\alpha_{pi}) = 0.01$

And then a second model more complex:

- Number of encoder/decoder layers = 4
- Number of attention heads = 8
- Model dimension $(d_{\text{model}}) = 100$
- Feed-forward network dimension $(d_{\text{ff}}) = 512$
- Dropout rate = 0.1
- Physics-informed loss penalty hyperparameter $(\alpha_{pi}) = 0.01$

2.5 Evaluation Metrics

The performance of both LSTM and Transformer hybrid models will be evaluated using a combination of data-driven and physics-informed metrics:

- **Mean Squared Error (MSE):** For observed variables, to quantify the data-driven prediction accuracy.
- **Probability Density Function (PDF) Comparison:** For unmeasured variables, comparing the PDFs of reconstructed trajectories with ground truth using metrics like Kullback-Leibler (KL) divergence. Q-Q plots will also be used to assess agreement, especially for tail behavior.
- **Lyapunov Exponents (LEs):** To assess the models' ability to reproduce the chaotic dynamics, comparing the spectrum of LEs $(\lambda_1, \ldots, \lambda_N)$ from the model's autonomous evolution against the true system's LEs. Relative errors in the leading LEs will be tabulated.
- **Long-term Prediction Horizon:** Evaluating how long the models can accurately predict the system's state in closed-loop mode before diverging significantly from the true trajectory.

2.6 Entropie Métrique de Kolmogorov exposant de Lyapunov

2.7 Results and Comparative Analysis

2.7.1 OpenLoopTest

Convergence: Training vs Validation Loss

The convergence behavior of both the PI-LSTM and the Transformer-based hybrid model during training will be analyzed by plotting their respective training and validation loss curves. These plots will illustrate the learning progress and indicate potential overfitting. Quantitative assessment of convergence can be achieved by computing the Area Under the Curve (AUC) for the validation loss, providing a single metric for overall training stability and performance.

2.7.2 ClosedLoopTest

PDF Comparison: Ground Truth vs LSTM

For the PI-LSTM, the long-term statistical behavior of the reconstructed unmeasured variables will be assessed by comparing their Probability Density Functions (PDFs) with those of the ground truth Lorenz-96 system. Histograms and Kernel Density Estimates (KDEs) will be used to visualize these distributions. The Kullback-Leibler (KL) divergence will quantify the difference between the reconstructed and target PDFs. Additionally, Q-Q plots will be employed to provide a visual comparison of the quantiles, highlighting any systematic biases or discrepancies in the distribution's shape, particularly in the tails.

PDF Comparison: Ground Truth vs Transformer

See also the property of ergodicity of the Lorenz-96 system, which implies that long-term statistical properties can be captured by a single trajectory. Similar to the LSTM analysis, the PDFs of the reconstructed unmeasured variables from the Transformer-based hybrid model will be compared against the ground truth. This will involve plotting histograms and KDEs, computing KL divergence, and using Q-Q plots. This comparative analysis will highlight the Transformer's ability to capture the long-term statistical properties of the chaotic system and identify any

systematic biases, such as under-dispersion or over-dispersion, especially at the extremes of the distribution.

Per-variable PDFs: LSTM (seqLen=200, epochs=40), run 2

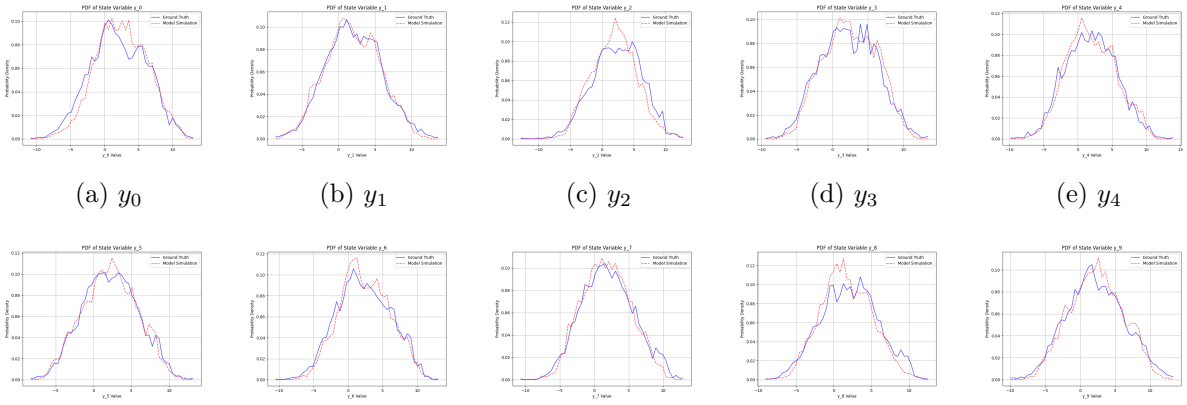
The following grid shows the per-variable probability density comparisons for the second run of the PI-LSTM with sequence length 200 and 40 epochs. All panels are arranged to fit on one page.

Lyapunov Exponent Analysis: Ground Truth vs LSTM vs Transformer

Compute the LE at each epochs taking the mean of the behaviour on each trajectory (possible efficiently ?) A critical evaluation will involve comparing the spectrum of Lyapunov exponents (λ_k) extracted from the autonomous closed-loop trajectories of the PI-LSTM and the Transformer-based model against the reference LEs of the true Lorenz-96 system. A table will present the values of $\lambda_1, \ldots, \lambda_N$ for all three cases, along with the relative errors for the leading positive LEs. Visualizing the convergence of the λ_1 estimate as a function of trajectory length will provide insight into the stability and accuracy of the chaotic dynamics reproduced by each model.

Tests with another trajectory

To further validate the generalization capabilities of the trained models, additional tests will be conducted using a different initial condition or a slightly perturbed trajectory from the Lorenz-96 system. This will assess the robustness of the PI-LSTM and Transformer models in reconstructing and predicting chaotic dynamics beyond the specific training trajectory. The same evaluation metrics (PDF comparison, LE analysis) will be applied to these new trajectories.



(a) y_0 (b) y_1 (c) y_2 (d) y_3 (e) y_4

(f) y_5 (g) y_6 (h) y_7 (i) y_8 (j) y_9

Figure 2.2: Per-variable PDF comparisons for PI-LSTM (seqLen=200, epochs=40), second run.

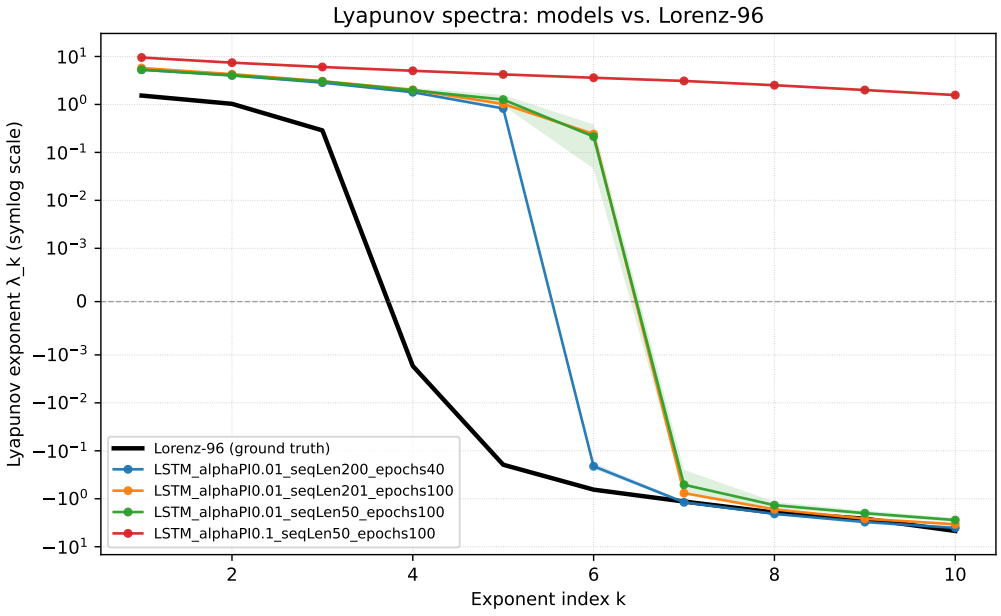


Figure 2.3: Lyapunov spectra comparison on a symmetric logarithmic vertical scale. The ground truth Lorenz-96 spectrum (black) is plotted against the averaged spectra of each trained model configuration.

Chapter 3

Discussion and Conclusions

3.1 Interpretability

The interpretability of PIML models, especially sequential ones, stems from their adherence to physical laws. By ensuring that the model's predictions satisfy known governing equations, the learned dynamics become more physically consistent and thus more interpretable. For instance, in chaotic systems, the ability of a PI-LSTM to accurately reproduce Lyapunov exponents, which characterize the system's chaoticity, provides a strong indicator of its physical fidelity. This contrasts with purely data-driven models that might achieve good short-term prediction but fail to capture the underlying long-term statistical and dynamical properties, leading to less interpretable or even misleading results.

3.2 Possible implementation

Il est rapide : une fois entraîné, il tourne en quelques millisecondes. On peut donc l'utiliser entre deux cycles d'assimilation (6h) pour : • prolonger l'état météo, • combler des trous, • générer des ensembles d'états réalistes.

3.3 Future Research Directions

3.4 Conclusions

3.5 Final Remarks about computability of chaotic systems using AI

One could think ai to be the best and last tool the humans will use. Just give it data and it will do the rest. Some people even think it will solve the problem of chaotic function. But let's step back for a moment.

One of the greatest aspects of Theory of information, computability and algorithmic theory of information, is their ability to give, guarantees, limits and trade-offs. Truth build by logic, limits inherent to every other concepts steming from them. Computers, numerical analysis, and AI are no exception.

Reecrire: One of the most fundamental of result of theory of information and computability is the undecidability of the halting problem. Some people showed that some choatic dynamical system were choatic beceause they are able to simulate turing machine and face then the same limitation , they sort of encode undecidability within themselves making impossible at some point to know things with certainty. I think what ai get is "the lower dimensional manifold" of what our eyes are limitted to see ...

Lot of people ignore the trade off complexity in size and complexity in time. Complexity in size refer to the Kolmogorov complexity (also called Solomonov complexity) which is a mesure of the size of the program solving a problem (here predict/reconstruct) and the time complexity which is the time it takes to run the program and output a good solution. Let's take the example of chess. The algorithm solving solving exactly the chess problem is the minimax algorithm. It is an algorithm with a small Kolmogorov complexity, it needs only a few lines of code to be implemented. But the time complexity is huge, it is exponential in the number of moves. One the other and, the AlphaZero algorithm is a neural network that learns to play chess. It has a huge Kolmogorov complexity because of all the number of parameters (weights and biases) it has plus the data used to train the model. But the time complexity is polynomial in the number of moves and it run in a reasonable time to make a move in a tournament. So here we trade complexity in size for complexity in time. At first glance it seems lika a good deal, but it would mean that in order to get faster and faster we would need to always increase the size of the model hence the size of the computers and the energy coming with it. It is faster to parallelize, but more costly. For all theses reasons , interpretability, objective and complexity, Ai must stay a tool. A tool of numerical analysis complementary to the previous tools and the knowledge build by physics and mathematics.

Bibliography

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] Wenyuan Li, Zili Liu, Keyan Chen, Hao Chen, Shunlin Liang, Zhengxia Zou, and Zhenwei Shi. Deepphysinet: Bridging deep learning and atmospheric physics for accurate and continuous weather modeling. *arXiv preprint arXiv:2401.04125*, 2024.

[3] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[4] Elise Özalp, Georgios Margazoglou, and Luca Magri. Physics-informed long short-term memory for forecasting and reconstruction of chaos. *arXiv preprint arXiv:2302.10779*, 2023.

Appendix A

Appendix

A.1 Notation

A.2 Additional Experiments