SLiDE: Structured Linear Dynamics for Forecasting with Exogenous Inputs

Sebastian Pütz *
Karlsruhe Institute of Technology,
Helmholtz AI

Theodore Glavas *
McGill University,
ILLS, Mila

Benjamin Schäfer
Karlsruhe Institute of Technology,
Helmholtz AI

Boris N. Oreshkin †
Amazon Science

Mark Coates
McGill University,
ILLS, Mila

Abstract

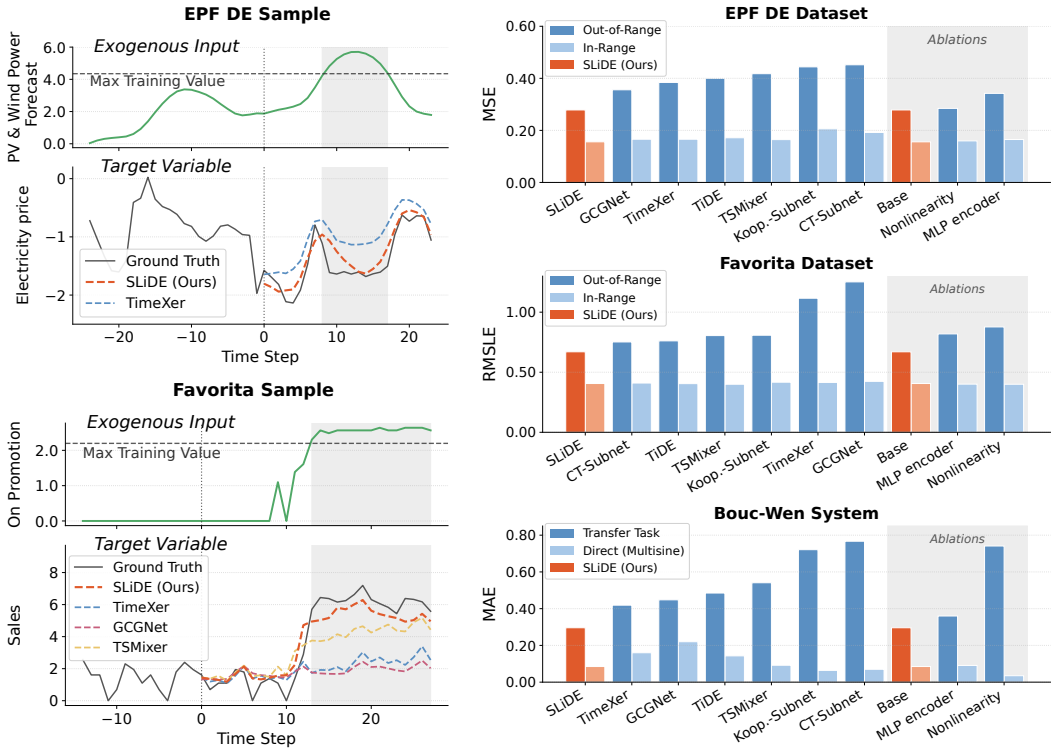
Accurate time series forecasting with exogenous inputs is critical across domains, including energy and retail, yet modern deep learning models often overfit to correlations that do not generalize under shifts in these inputs. We propose SLiDE, a Koopman-inspired architecture for forecasting with exogenous inputs that imposes structure on latent temporal dynamics. SLiDE approximates nonlinear system evolution using a learned linear operator in a latent space, combining (i) a history encoder that reconstructs the current latent state from past targets and inputs with (ii) a shared linear rollout driven by future exogenous variables. By enforcing a common linear recurrence across encoding and prediction, SLiDE aligns how past information is encoded with how future states evolve, reducing parameterization and preventing reliance on spurious correlations in the input history. Empirically, SLiDE achieves state-of-the-art accuracy on real-world benchmarks, including electricity price forecasting and retail demand, while maintaining a computational footprint comparable to lightweight MLP architectures. SLiDE is especially effective at generalizing to shifted exogenous inputs, reducing MSE by 20% on electricity price samples with out-of-range exogenous values and MAE by 30% in a synthetic transfer setting where the same dynamical system is driven by an unseen exogenous process. Ablations confirm the importance of both the structured history encoder and the linear latent rollout, suggesting that structured latent linear dynamics provide a useful inductive bias for forecasting with exogenous inputs.

1 Introduction

Time series forecasting supports decision-making across energy [Hong et al., 2020], economics [Stock and W. Watson, 2003], healthcare [Harutyunyan et al., 2019], transportation [Vlahogianni et al., 2014], and retail. In many settings, accurate forecasts depend on exogenous inputs [Box et al., 2015]: electricity prices depend on load and renewable generation forecasts [Weron, 2014, Trebbien et al., 2023], while retail demand depends on promotions, holidays, and product- or store-level context [Fildes et al., 2022]. These inputs are often forecasted, noisy, or generated by changing external processes, making robustness to shifted exogenous inputs important for deployment. Modern deep forecasting models incorporate exogenous information through cross-attention [Wang et al., 2024], graph encoding [Li et al., 2026], or MLP feature stacking [Das et al., 2023]. While effective,

*Equal contribution.
†This work is not related to the author's position at Amazon.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Test samples with out-of-range inputs

(b) Performance under shifted exogenous inputs

Figure 1: SLiDE generalizes well to shifted exogenous inputs, including out-of-range (OOR) input values and unseen test-time input processes. Left: top, an EPF-DE sample in which the wind forecast exceeds the maximum training value, and bottom, a Favorita sample in which the number of related promoted items (log-scale) exceeds the training range. In both cases, SLiDE stays closer to the ground truth than the baselines. Right: SLiDE reduces MSE by 20% on OOR EPF-DE samples and MAE by 30% on the Bouc-Wen transfer task, where models are tested on an unseen forcing process. Ablations show that the structured encoder and linear rollout are critical for shifted-input performance, as shown by an 18% lower RMSLE relative to ablations on OOR Favorita samples.

they typically rely on flexible feature fusion rather than structured input-driven temporal evolution. This can let models exploit input-history correlations that do not reflect the underlying input-response behavior. Attention and graph modules add computational overhead, while lightweight MLPs may lack temporal structure.

Koopman-inspired modeling provides a useful architectural bias. For autonomous nonlinear systems, Koopman theory represents nonlinear evolution as linear dynamics over *observables*, i.e., scalar-valued functions of the state. Learning nonlinear lifting with linear latent transport may be easier to learn than a nonlinear latent transition directly. Recent applications such as Zhang et al. [2025] suggest that this provides a suitable bias for the forecasting setting.

Motivated by this, we propose SLiDE (**S**tructured **Li**near **D**ynamics with **E**xogenous inputs), a lightweight Koopman-inspired architecture for multi-step forecasting with exogenous inputs. Rather than implementing an exact finite-dimensional Koopman model, SLiDE uses learned latent linear evolution as a practical forecasting bias. It reconstructs a forecast-start latent state from past targets and inputs, then rolls it forward with future-known exogenous variables through two constraints: a lag-invariant recurrent encoder and a shared linear rollout with additive exogenous forcing.

SLiDE achieves strong forecasting performance while remaining computationally efficient. Across electricity price and retail demand benchmarks, it attains the best average rank with cost comparable to lightweight MLP architectures. Its advantage is clearest under shifted exogenous inputs, including out-of-range (OOR) values and unseen test-time input processes. Figure 1a shows shifted-input examples from EPF-DE and Favorita. In EPF-DE, the future wind forecast falls outside the training

range, and SLiDE tracks the ground-truth price while the baseline deviates. In Favorita, the number of promoted items is OOR, and SLiDE better tracks sales. Figure 1b shows that SLiDE reduces Mean Squared Error (MSE) by 20% on OOR EPF-DE samples and Mean Absolute Error (MAE) by 30% in Bouc-Wen transfer, where the same nonlinear system is driven by an unseen forcing process at test time. SLiDE achieves an 18% lower Root Mean Squared Logarithmic Error (RMSLE) than ablations without the structured encoder or linear latent rollout on OOR Favorita samples.

Our contributions are as follows: We introduce SLiDE, a lightweight architecture for multi-step forecasting with exogenous inputs that replaces unconstrained feature fusion with a Koopman-inspired latent recurrence. Its key constraints are a lag-invariant recurrent encoder for forecast-start state reconstruction and a shared linear rollout with additive exogenous forcing. Empirically, SLiDE achieves state-of-the-art performance across electricity price, retail demand, and nonlinear dynamical-system benchmarks, with the largest gains under shifted exogenous inputs. The code is publicly available.[3]

2 Related work

Deep learning has become a dominant paradigm for time series forecasting, and due to its practical importance, incorporating exogenous variables is an active area of research. Approaches to forecasting with exogenous variables include attention-based architectures such as TFT and TimeXer [Lim et al., 2021, Wang et al., 2024], lightweight feedforward models such as TiDE, TSMixer, and N-BEATSx [Das et al., 2023, Chen et al., 2023, Olivares et al., 2023, Oreshkin et al., 2020], and graph-based models such as GCGNet [Li et al., 2026]. In contrast to attention-, graph-, and MLP-based approaches that primarily rely on flexible feature fusion, SLiDE imposes a constrained input-driven latent dynamics model, which we show improves accuracy and efficiency while promoting generalization under shifted input processes.

A related line of work studies structured latent models with inputs from the perspective of nonlinear system identification and control, including Koopman-Subnet [Iacob et al., 2021], CT-Subnet [Beintema et al., 2023], and MamKO [Li et al., 2025], the latter combining Koopman-style latent operators with Mamba-based sequence modeling [Gu and Dao, 2024]. These models are related in spirit to our work in that they impose structure on latent dynamics, but they are primarily motivated by modeling, identification, or control rather than exogenous multi-step forecasting. SLiDE deviates from exact Koopman theory where necessary, incorporating deliberate design choices that make the technique specifically suitable for forecasting with known future exogenous inputs.

More broadly, SLiDE is related to Koopman-with-input works [Proctor et al., 2018, Korda and Mezić, 2018], Koopman-inspired forecasting models that do not incorporate exogenous variables [Zhang et al., 2025, Liu et al., 2023], and structured state space models such as S4 and Mamba [Gu et al., 2022, Gu and Dao, 2024]. Additional discussion of related work is provided in Appendix I.

3 Problem setting

We consider a discrete-time target series of real-valued observations $\boldsymbol{y}_\tau \in \mathbb{R}^{n_y}$. At forecast origin k, we observe the past target trajectory $\boldsymbol{y}_{[k]} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_k]$, where $[k]$ denotes the index set $[1, \dots, k]$, and the past exogenous trajectory $\boldsymbol{u}_{[k]} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_k]$ for an input $\boldsymbol{u}_\tau \in \mathbb{R}^{n_u}$ that affects the target but is not affected by it. We also assume access to the values available at time k for the next H forecast steps, denoted by $\boldsymbol{u}_{k,[H]} = [\boldsymbol{u}_{k,1}, \dots, \boldsymbol{u}_{k,H}]$. The double index indicates forecast origin k and lead time $h \in [H]$. These future exogenous values may themselves be forecasts and need not coincide with the values that are later observed. For example, the value available at origin k for lead time two, $\boldsymbol{u}_{k,2}$, may differ from both the realized value $\boldsymbol{u}_{k+2}$ and from a later updated forecast $\boldsymbol{u}_{k+1,1}$.

The forecasting task is to generate a point estimate from the conditional distribution of the target random variable $\mathbf{Y}_{k,[H]} = [\mathbf{Y}_{k,1}, \dots, \mathbf{Y}_{k,H}] = [\mathbf{Y}_{k+1}, \dots, \mathbf{Y}_{k+H}]$:

$$\mathbb{P}\left(\mathbf{Y}_{k,[H]} \mid \theta, \boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]}, \boldsymbol{u}_{k,[H]}\right), \tag{1}$$

[3] `https://github.com/KIT-IAI-DRACOS/slide`.

where θ are the model parameters. Our desired point forecast of the target is denoted by $\hat{\boldsymbol{y}}_{k,[H]}$.

Given D training samples of the form $(\boldsymbol{y}_{[k+H]}, \boldsymbol{u}_{[k]}, \boldsymbol{u}_{k,[H]})$, our goal is to learn the model parameters by minimizing a task-dependent risk $\mathcal{R}(\boldsymbol{y}_{k,[H]}, \hat{\boldsymbol{y}}_{k,[H]})$. The exact risks are given in Appendix F.1.

4 Background: Koopman theory

We model the observed time-series as a nonlinear system with state variable $\boldsymbol{x} \in \mathbb{R}^{n_x}$ at time index τ, and a nonlinear transition map $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$:

$$\boldsymbol{x}_{\tau+1} = f(\boldsymbol{x}_\tau, \boldsymbol{u}_{\tau+1}) \,, \tag{2a}$$

$$\boldsymbol{y}_\tau = o(\boldsymbol{x}_\tau) \,. \tag{2b}$$

We assume that the model state is (in general) not fully observable. Instead, we observe an output $\boldsymbol{y} \in \mathbb{R}^{n_y}$ obtained from a nonlinear output map $o : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$.

The Koopman operator framework [Koopman, 1931, Brunton et al., 2022] provides a means of representing nonlinear dynamical systems as evolving linearly in an infinite-dimensional lifted state. Let $\phi : \mathbb{R}^{n_x} \to \mathbb{R}$ be a scalar measurement function that is an element of an infinite-dimensional Hilbert space $\mathcal{H}$. For the corresponding zero-input autonomous system, define $f_0(\boldsymbol{x}) := f(\boldsymbol{x}, 0)$. The linear infinite-dimensional Koopman operator $\mathcal{K} : \mathcal{H} \to \mathcal{H}$ acts on the measurement functions and satisfies

$$\mathcal{K}\phi = \phi \circ f_0, \quad \forall \phi \in \mathcal{H} \,, \tag{3}$$

or equivalently $\mathcal{K}\phi(\boldsymbol{x}_\tau) = \phi(f_0(\boldsymbol{x}_\tau))$. In practice, we approximate $\mathcal{H}$ by a finite dimensional subspace $\mathcal{H}_{n_z} \subset \mathcal{H}$ with $\mathcal{H}_{n_z} = \operatorname{span}\{\phi_j\}_{j=1}^{n_z}$ and associated approximate operator $\mathcal{K}_{n_z}$.

When there are inputs $\boldsymbol{u}$, it becomes more complicated to preserve the linearity inherent to the Koopman framework. Following Iacob et al. [2024], we introduce the concept of a lifted state $\boldsymbol{z} \in \mathbb{R}^{n_z}$ where $\boldsymbol{z}_\tau = \Phi(\boldsymbol{x}_\tau) = [\phi_1(\boldsymbol{x}_\tau), \ldots, \phi_{n_z}(\boldsymbol{x}_\tau)]$. We assume that $\Phi \circ f_0(\cdot) \in \operatorname{span}\{\Phi\}$, implying that there exists a matrix $A \in \mathbb{R}^{n_z \times n_z}$ such that

$$\Phi \circ f_0 = A\Phi. \tag{4}$$

We also assume that our output map o can be exactly represented by a linear combination of the observables Φ, so:

$$o(\cdot) = C\Phi(\cdot) \,, \quad \text{with } C \in \mathbb{R}^{n_y \times n_z} \,. \tag{5}$$

While this can be achieved by including the output functions among the observables, requiring their span to also be Koopman-invariant is generally restrictive in finite dimensions. Under these assumptions, Iacob et al. [2025] show that there exists an exact finite-dimensional Koopman form

$$\boldsymbol{z}_{\tau+1} = A\boldsymbol{z}_\tau + B(\boldsymbol{z}_\tau, \boldsymbol{u}_{\tau+1})\boldsymbol{u}_{\tau+1} \tag{6}$$

where $A \in \mathbb{R}^{n_z \times n_z}$ and $B(\boldsymbol{z}_\tau, \boldsymbol{u}_{\tau+1})$ is, in general, a nonlinear state- and input-dependent map. SLiDE uses the input-dependent Koopman form as architectural motivation and adopts a practical forecasting-oriented parameterization of the input effect.

5 Methodology

In SLiDE, a structured encoder maps past targets and exogenous inputs to a forecast-start latent state, which a shared linear operator W rolls forward using additive forcing from future-known inputs (see Figure 2). A nonlinear output map maps the latent trajectory back to the target space, and multiple block forecasts are summed. We now proceed with the design principles, first motivating them based on Koopman theory, and then introducing the important adjustments for forecasting success.

5.1 Koopman-inspired forecasting formulation

Following Iacob et al. [2025] we can write our model as

$$\boldsymbol{z}_{\tau+1} = A\boldsymbol{z}_\tau + \mathcal{B}\boldsymbol{u}_{\tau+1} \,, \tag{7a}$$

$$\boldsymbol{y}_\tau = C\boldsymbol{z}_\tau \,. \tag{7b}$$

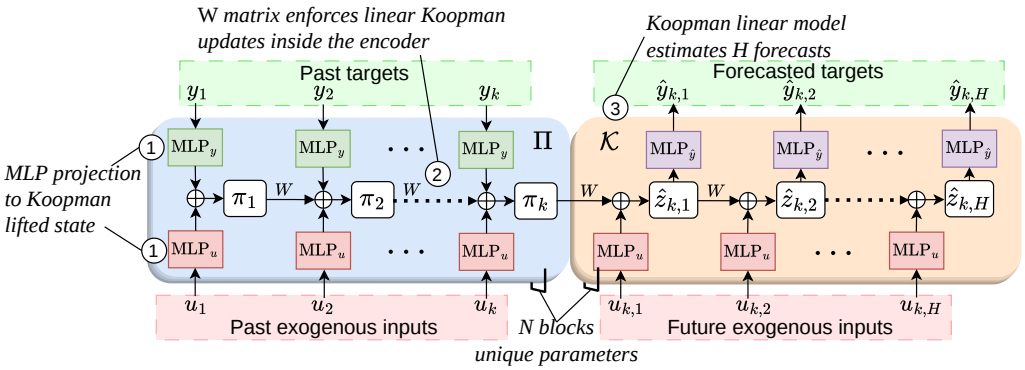


Figure 2: SLiDE Architecture: (1) MLPs project past targets and exogenous inputs to a series of hidden states $\pi_1, \ldots, \pi_k$ inside the encoder Π. (2) The encoder is parametrized by a matrix W, which is applied recurrently to impose linear latent dynamics. The encoder estimates the present lifted state $\pi_k = \hat{\boldsymbol{z}}_k$. (3) The prediction module $\mathcal{K}$ uses $\hat{\boldsymbol{z}}_k$ and future exogenous inputs to forecast the next H targets. It shares the same matrix W and exogenous input projection map, with an additional MLP to project back to the target dimension. N independently parametrized SLiDE blocks are jointly trained to learn complementary dynamics. The final output is taken as the sum of block forecasts.

Here the matrix-valued input function $\mathcal{B}$ may be chosen simpler than in the exact formulation $B(\boldsymbol{z}_\tau, \boldsymbol{u}_{\tau+1})$ to trade off accuracy with model simplicity e.g. by choosing a linear, bilinear or input affine model structure [Iacob et al., 2025]. Allowing an unrestricted lifted state dependence in $\mathcal{B}$ could introduce nonlinear dynamics in the lifted state via the input term. In our case we restrict the input function $\mathcal{B}$ to be independent of the lifted state (Appendix B.1 provides further context on the implications of our choice). We use this model as a motivation and treat it as a structural prior for forecasting rather than aiming at finding an exact Koopman realization.

One challenge of implementing this model is determining the initial condition of the lifted state $\boldsymbol{z}_k$. Iacob et al. [2025] introduce a reconstructability concept and discuss its relationship with local observability. The *observability map* $\Lambda_{\boldsymbol{z},m}$ can be used to compute the lifted state at the start of a window from m consecutive output values and the corresponding inputs. Relying on this, a *reconstructability map* $\Pi_{\boldsymbol{z},k}$ can be derived to determine the initial condition $\boldsymbol{z}_k$ from k *past* (lagged) outputs $\boldsymbol{y}_{[k]}$ and the $k-1$ inputs that follow the first observation, given sufficiently large k. We provide the derivation in the context of our method in Appendix A.

In order to estimate this model in practice, we parameterize $A_\theta, \mathcal{B}_\theta, C_\theta$. We collect these parameters in $\theta \in \Theta \subseteq \mathbb{R}^{n_\theta}$. Since an analytical solution for $\Pi_{\boldsymbol{z},k}$ is impractical, we introduce a learned approximation $\Pi^{\psi}_{\boldsymbol{z},k}$ with parameters $\psi \in \Psi \subseteq \mathbb{R}^{n_\psi}$, which additionally receives $\boldsymbol{u}_1$ for alignment (Appendix A). All parameters are learned jointly with the goal of minimizing a general risk function $\mathcal{R}(\boldsymbol{y}_{k,[H]}, \hat{\boldsymbol{y}}_{k,[H]})$.

In practice, the future values $\boldsymbol{u}_{k,h}$ available at forecast origin k allow us to estimate future lifted states $\hat{\boldsymbol{z}}_{k,h}$ using the same indexing. Our estimation approach for multi-step forecasting is then:

$$\hat{\boldsymbol{z}}_k = \Pi^{\psi}_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]})\,, \tag{8a}$$

$$\hat{\boldsymbol{z}}_{k,h} = A_\theta \hat{\boldsymbol{z}}_{k,h-1} + \mathcal{B}_\theta \boldsymbol{u}_{k,h}\,, \qquad \text{with initialization } \hat{\boldsymbol{z}}_{k,0} := \hat{\boldsymbol{z}}_k\,, \tag{8b}$$

$$\hat{\boldsymbol{y}}_{k,h} = C_\theta \hat{\boldsymbol{z}}_{k,h},\ h \in \{1, \ldots, H\}\,. \tag{8c}$$

5.2 Defining a structure on the reconstructability map

Beyond establishing its theoretical existence, prior work gives little guidance on the practical form of $\Pi_{\boldsymbol{z},k}$ and typically treats it as a generic black-box encoder, implemented as a dense MLP learned directly from data [Iacob et al., 2025]. For forecasting, this approach has a number of practical challenges: a dense MLP ignores the sequential structure of the past trajectory, its parameterization grows with the available context length, and its computational cost can dominate the relatively

lightweight latent rollout. We therefore impose additional structure on the learned encoder $\Pi^{\psi}_{\boldsymbol{z},k}$, introducing a forecasting-oriented inductive bias for reconstructing the forecast-start latent state:

Lag-invariant parameterization. We constrain $\Pi^{\psi}_{\boldsymbol{z},k}$ to have the same parameterization $\psi \in \Psi \subseteq \mathbb{R}^{n_\psi}$ for any positive k. The parameter search space Ψ is fixed as a design choice and does not grow with the number of lagged values k. This prevents the model search space from exploding and reduces model memory cost. Furthermore, it allows for a single ψ to be learned from training samples with varying amounts of past data available.

Encoder Koopman update. The exact reconstructability map is recursively related to the latent dynamics: reconstructing $\boldsymbol{z}_k$ from history can be viewed, informally, as reconstructing an earlier latent state and propagating it forward. Motivated by this structure, we impose a recurrent latent update inside the encoder. In particular, we define

$$\Pi^{\psi}_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]}) = \hat{\boldsymbol{z}}_k \tag{9a}$$

$$= A_\psi \hat{\boldsymbol{z}}_{k-1} + \mathcal{B}_\psi \boldsymbol{u}_k + \mathcal{D}_\psi \boldsymbol{y}_k \tag{9b}$$

$$= A_\psi \Pi^{\psi}_{\boldsymbol{z},k-1}(\boldsymbol{y}_{[k-1]}, \boldsymbol{u}_{[k-1]}) + \mathcal{B}_\psi \boldsymbol{u}_k + \mathcal{D}_\psi \boldsymbol{y}_k \tag{9c}$$

for some linear matrix A_ψ and MLPs $\mathcal{B}_\psi$ and $\mathcal{D}_\psi$. This recurrence requires the encoder to learn not just a final estimate $\hat{\boldsymbol{z}}_k$, but a trajectory of estimates that is updated with the same linear structure as the underlying Koopman system. In this context, $\boldsymbol{y}$ is modeled as an additive input term alongside $\boldsymbol{u}$, helping to provide the information required to guide the trajectory of $\hat{\boldsymbol{z}}$ towards a final estimate $\hat{\boldsymbol{z}}_k$.

Parameter sharing. In order to limit the parameterization of our model and align the dynamics inside the encoder with those used in the forecast horizon, we impose complete parameter sharing by setting $A_\psi = A_\theta$ and $\mathcal{B}_\psi = \mathcal{B}_\theta$. This imposes a strong heuristic for the encoder to follow the dynamics of the Koopman model. However, the inclusion of the $\mathcal{D}_\psi \boldsymbol{y}_k$ term allows for some differences, so that the known lagged values $\boldsymbol{y}_{[k]}$ can influence the reconstruction of $\hat{\boldsymbol{z}}_k$.

5.3 Model definition

Pre-processing: Patching. For long input histories or forecast horizons, we may optionally apply non-overlapping patching to the target and input sequences, reducing the effective sequence length while increasing the per-step target and input dimensions. The exact transformation is given in Appendix B. After patching, we reuse the notation $\boldsymbol{y}$ and $\boldsymbol{u}$ for the transformed target and input sequences.

Implementation of the Koopman operator. We implement the Koopman operator approximation as a matrix $A_\theta = A_\psi := W \in \mathbb{R}^{n_z \times n_z}$, with parameter sharing as imposed in Section 5.2. For the input maps $\mathcal{B}_\theta = \mathcal{B}_\psi$ and $\mathcal{D}_\psi$, we approximate the terms

$$\mathcal{B}_\theta \boldsymbol{u}_\tau \approx \text{MLP}_{\boldsymbol{u}}(\boldsymbol{u}_\tau) = \mathbf{p}_\tau \tag{10a}$$

$$\mathcal{D}_\psi \boldsymbol{y}_\tau \approx \text{MLP}_{\boldsymbol{y}}(\boldsymbol{y}_\tau) = \mathbf{d}_\tau \tag{10b}$$

The MLPs have much more representational ability than a fixed matrix but are less general than $B(\boldsymbol{z}_\tau, \boldsymbol{u}_\tau)\boldsymbol{u}_\tau$, since the MLP output does not vary with $\hat{\boldsymbol{z}}_\tau$. In time series forecasting tasks, the external input provided to the model may be noisy or uncertain. This choice provides a practical compromise. By alleviating restrictions on the input term, we can better match the complexity of real-world data. The resulting forecast rollout then becomes

$$\hat{\boldsymbol{z}}_{k,h} = W \hat{\boldsymbol{z}}_{k,h-1} + \mathbf{p}_{k,h}, \quad \text{for } h \in \{1, \ldots, H\}. \tag{11}$$

Definition of the output map. Although the idealized formulation uses $\hat{\boldsymbol{y}}_{k,h} = C_\theta \hat{\boldsymbol{z}}_{k,h}$, we implement the output map as the nonlinear function $\text{MLP}_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{z}}_{k,h})$. This added expressivity helps compensate for approximation error in the Koopman-inspired model.

Definition of Encoder $\Pi^{\psi}_{\boldsymbol{z},k}$. We encode the initial current lifted state $\boldsymbol{z}_k$ from the k lagged values of the past trajectory. Using the previously defined $\text{MLP}_{\boldsymbol{u}}(\boldsymbol{u}_\tau) = \mathbf{p}_\tau$ and $\text{MLP}_{\boldsymbol{y}}(\boldsymbol{y}_\tau) = \mathbf{d}_\tau$, we define the encoder $\Pi^{\psi}_{\boldsymbol{z},k}$ using a linear recurrent neural network with a hidden state $\boldsymbol{\pi} \in \mathbb{R}^{n_z}$, where

$$\boldsymbol{\pi}_0 = \mathbf{0}, \qquad \boldsymbol{\pi}_\tau = W \boldsymbol{\pi}_{\tau-1} + \mathbf{d}_\tau + \mathbf{p}_\tau, \quad \tau \in \{1, \ldots, k\}, \tag{12a}$$

$$\Pi^{\psi}_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]}) = \hat{\boldsymbol{z}}_k := \boldsymbol{\pi}_k \,. \tag{12b}$$

The equality $\hat{\boldsymbol{z}}_k = \boldsymbol{\pi}_k$ only holds at time k. We can write the closed-form equation for computing $\hat{\boldsymbol{z}}$ as repeated applications of the matrix W on the encoded $\boldsymbol{y}$ and $\boldsymbol{u}$ lags:

$$\Pi^{\psi}_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]}) = \hat{\mathbf{z}}_k := \mathbf{d}_k + \mathbf{p}_k + \sum_{\tau=1}^{k-1} (W)^{\tau}(\mathbf{d}_{k-\tau} + \mathbf{p}_{k-\tau}) \tag{13}$$

Block structure. We impose a block structure on the entire forecasting model by decomposing the lifted state into N non-interacting subspaces. Each SLiDE block $b \in \{1, \ldots, N\}$ is intended to capture different dynamics of the target system. We consider our full lifted state $\boldsymbol{z}$ to be separable into N sub-states $\boldsymbol{z}^b \in \mathbb{R}^{n_{z,N}}$ where $\boldsymbol{z} = [{\boldsymbol{z}^1}^\top, \ldots, {\boldsymbol{z}^N}^\top]^\top$ and $n_{z,N} = \frac{n_z}{N}$. Each subspace is initialized by a separate encoder $\Pi^{\psi}_{\boldsymbol{z}^b,k}$. We enforce a block-diagonal structure: $A_\theta = \mathrm{diag}(A_{\theta,1}, \ldots, A_{\theta,N})$, which allows $\boldsymbol{z}^b$ to evolve independently through time. A separate output MLP is used to obtain $\hat{\boldsymbol{y}}^b_{k,[H]}$, with the block forecasts summed to obtain the final output: $\hat{\boldsymbol{y}}_{k,[H]} = \sum_{b=1}^{N} \hat{\boldsymbol{y}}^b_{k,[H]}$.

6 Experimental results

In this section, we evaluate SLiDE along three axes: *(1) Forecasting performance* on real-world and synthetic benchmarks; *(2) Generalization under shifted exogenous inputs*, including out-of-range real-world samples and the Bouc–Wen transfer setting; and *(3) Efficiency*, assessing the tradeoff between predictive performance and computational cost.

Benchmarks. We evaluate SLiDE on real-world datasets with heterogeneous covariates, including future-known, past-only, and static variables. In SLiDE, future-known covariates are treated as exogenous inputs $\boldsymbol{u}$, past-only covariates are modeled jointly with the target history, and static covariates are embedded and concatenated repeatedly to the input signal $\boldsymbol{u}$.

The *Electricity Price Forecasting (EPF) benchmark* [Lago et al., 2021] consists of five datasets from different electricity markets and is a standard benchmark for forecasting with future-known covariates [Li et al., 2026, Wang et al., 2024, Olivares et al., 2023]. The *Favorita sales dataset* [Cook et al., 2021] targets large-scale retail demand forecasting with promotions and holidays known in advance. We additionally consider the synthetic *Bouc-Wen hysteretic system* [Schoukens and Noël, 2017, Noël and Schoukens, 2020], a nonlinear dynamical model with exogenous forcing. We use it both as a controlled forecasting task and as a test of generalization under shifted input processes. In the *Multisine* setting, models are trained and evaluated on a random-phase multisine forcing trajectory. In the *Transfer* setting, models are trained on a sine-sweep forcing process and evaluated on a multisine forcing process at test time. Appendix F.1 provides additional dataset details.

Metrics. Performance is measured by Mean Absolute Error (MAE) and Mean Squared Error (MSE) for the EPF benchmark and the Bouc-Wen system. For the Favorita sales dataset, we report the Root Mean Squared Logarithmic Error (RMSLE). To assess statistical significance, we perform pairwise one-sided tests on forecast-error differences between models. We use the Diebold–Mariano test for all datasets except Favorita, for which we use a Wilcoxon test on per-series error differences. Appendix H provides additional details and bootstrapped confidence intervals. To compare computational cost, we report the mean inference time and the trainable parameter count for each model.

Baselines. We compare SLiDE against two baseline groups: competitive forecasting architectures that support exogenous information, and related Koopman- or system-identification-inspired models. The forecasting baselines are TSMixer [Chen et al., 2023], TiDE [Das et al., 2023], TimeXer [Wang et al., 2024], GCGNet [Li et al., 2026], and DLinear [Zeng et al., 2023] as a simple linear baseline. We also include SKOLR [Zhang et al., 2025], a related Koopman-inspired model that cannot use future-known covariates to show the performance lift from auxiliary covariates. To position SLiDE relative to structured latent-dynamics approaches, we compare against Koopman-Subnet [Iacob et al., 2021], CT-Subnet [Beintema et al., 2023], and MamKO [Li et al., 2025], which are conceptually relevant Koopman-with-input or system-identification baselines, though not originally proposed for the exogenous forecasting setup studied here. Appendix F.2 provides additional context.

Training Methodology. To ensure a fair comparison, we select the hyperparameters for all baselines using the same random-search protocol with a fixed budget and patience. For each model, we select the hyperparameter configuration that achieves the lowest validation loss. Each model is then trained five times with different random seeds and the results are averaged. Additional details on the search spaces and training settings are provided in Appendix F.2.

Table 1: Performance on real-world and synthetic benchmarks, including the main ablations. The reported metrics are averaged across five random seeds. We highlight the **first**, second and third results for each metric across baseline comparisons and the **first** across ablations. The rightmost column reports the average rank across metrics for the baselines and a hypothetical ranking for ablations. SLiDE achieves the best average rank overall, and the ablations show that replacing the structured encoder or the linear latent rollout degrades performance.

Model	EPF Benchmark										Favorita	Bouc-Wen System				Avg.
	NP		PJM		EPEX BE		EPEX FR		EPEX DE		Sales	Multisine		Transfer		Rank
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	RMSLE	MAE	MSE	MAE	MSE	
SLiDE	**0.210**	**0.167**	0.157	0.064	0.231	0.367	**0.169**	0.329	**0.265**	**0.182**	**0.419**	0.085	0.012	**0.296**	**0.139**	1.87
Forecasting Architectures:																
TimeXer	0.220	0.186	**0.154**	**0.060**	**0.230**	0.367	**0.169**	0.347	0.278	0.213	0.461	0.160	0.040	0.419	0.273	3.27
TSMixer	0.219	0.184	0.158	0.063	0.235	**0.335**	0.189	0.331	0.283	0.218	0.422	0.092	0.013	0.541	0.480	3.27
GCGNet	0.220	0.177	0.157	0.065	0.252	0.367	0.189	**0.320**	0.285	0.206	0.482	0.220	0.086	0.448	0.322	3.87
TiDE	0.243	0.200	0.165	0.081	0.256	0.362	0.200	0.353	0.295	0.220	0.424	0.143	0.031	0.485	0.385	5.27
DLinear	0.249	0.211	0.166	0.074	0.258	0.371	0.198	0.363	0.307	0.248	0.483	0.793	0.966	1.354	2.919	7.93
SKOLR	0.264	0.237	0.185	0.097	0.256	0.405	0.201	0.372	0.410	0.432	0.478	0.754	0.862	0.782	0.929	9.00
Related System Identification or Control Baselines:																
Koop.-Subnet	0.251	0.214	0.181	0.089	0.250	0.360	0.196	0.345	0.312	0.257	0.438	**0.064**	**0.007**	0.721	0.815	5.80
CT-Subnet	0.237	0.185	0.175	0.082	0.260	0.391	0.208	0.375	0.318	0.247	0.428	0.070	0.010	0.767	0.885	6.67
MamKO	0.245	0.213	0.172	0.079	0.256	0.374	0.192	0.324	0.325	0.276	0.489	0.464	0.340	0.944	1.722	7.40
Ablations:																
Base (SLiDE)	**0.210**	**0.167**	**0.157**	**0.064**	**0.231**	0.367	**0.169**	**0.329**	**0.265**	**0.182**	**0.419**	0.085	0.012	**0.296**	**0.139**	1.87
Nonlinearity	0.214	0.170	0.158	0.067	0.236	0.371	**0.169**	**0.329**	0.269	0.186	0.428	**0.035**	**0.002**	0.741	0.877	2.73
MLP encoder	0.223	0.172	0.160	0.065	0.239	**0.352**	0.183	0.332	0.281	0.203	0.423	0.091	0.014	0.360	0.207	2.47

6.1 Key results

Accuracy. SLiDE is a consistent top performer on the real-world datasets, ranking among the top two in eight out of eleven instances. On the synthetic Bouc-Wen benchmark, it achieves the strongest overall performance across forecasting architectures in the multisine task and the strongest performance overall in the transfer setting. While the related system identification and control baselines achieve top performance on the Bouc-Wen multisine task, they are not competitive with the forecasting architectures in general, achieving at best an average rank of 5.80. We report the benchmark results in Table 1. Overall, SLiDE significantly outperforms each baseline multiple times while being significantly outperformed at most twice (see Figure 3a).

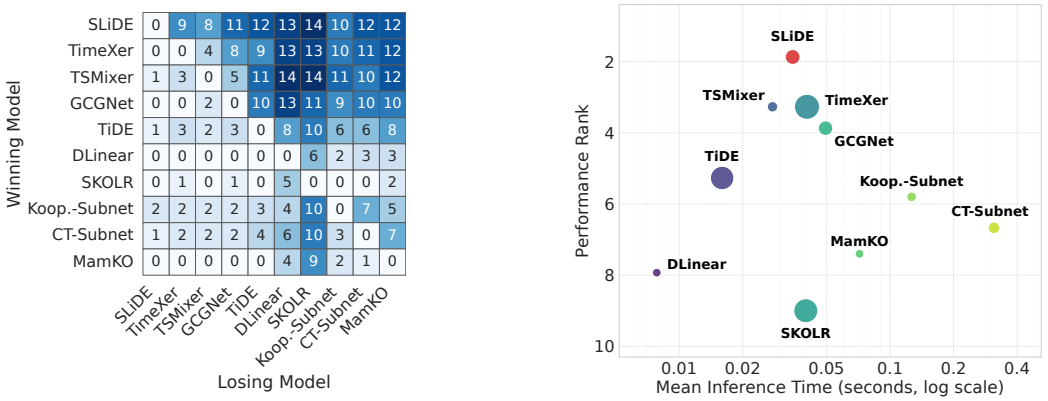
Shifted Exogenous Inputs. SLiDE performs particularly well under shifts in the exogenous inputs. On the real-world side, we study *out-of-range* (OOR) exogenous inputs, defined as test samples for which at least one future-known exogenous value lies outside the range observed during training. We focus our analysis on EPF-DE as it contains the largest number of such samples among the EPF datasets. Figure 1b summarizes the aggregate results: on OOR EPF-DE samples, SLiDE reduces MSE by 20% relative to the strongest baseline. We observe a similar pattern on Favorita, where we observe a clear separation from the baselines on a small subset of OOR samples (see Appendix C). On the synthetic side, we use the Bouc-Wen transfer setting, where the test-time input process differs from training. In this setting, SLiDE reduces MAE by 30% relative to the next best model. These results support the view that SLiDE generalizes better under shifted exogenous inputs.

Efficiency. We compare mean inference time against average benchmark rank in Figure 3b. SLiDE has inference cost similar to TSMixer, a model designed for computational efficiency, while being considerably faster than TimeXer and GCGNet. Marker size reflects the number of trainable parameters. Thus, SLiDE's superior performance does not come at a substantial computational cost.

6.2 Ablation studies

We perform targeted ablation experiments to assess the contribution of the main structural components of SLiDE. In particular, we study the role of the structured encoder and the linear latent rollout. For each ablation, we perform a random search over hyperparameters with fixed budget.

Nonlinear State Update. Our proposed architecture deliberately enforces a linear temporal evolution in latent space. To examine whether this provides a useful inductive bias, we perform an ablation that



(a) Pairwise significance results.

(b) Efficiency-performance tradeoff.

Figure 3: **(a)** Pairwise statistical test results across benchmark–metric combinations. Each cell reports how often the model on the row significantly outperforms the model on the column. **(b)** Mean inference time versus average performance rank; marker size is proportional to the number of trainable parameters. SLiDE achieves a notably higher ranking without compromising on computational cost.

introduces a nonlinear transformation into the latent dynamics by replacing W with a two-layer MLP with ReLU activation. We sweep over different hidden dimensions to vary model complexity.

MLP Encoder. We relax the constraints on the encoder $\Pi^{\psi}_{\boldsymbol{z},k}$ by replacing it with a dense MLP while keeping the shared latent rollout unchanged. We preserve the multiple-block structure and tune the hidden dimension and number of hidden layers as hyperparameters.

Ablation results. Table 1 reports the performance of both ablations across all datasets. Replacing the structured encoder with a dense MLP generally degrades performance, indicating that the lag-invariant recurrent encoder is an important component of the model. The effect of introducing a nonlinearity into the latent rollout is more nuanced. On the real-world benchmarks, it changes performance only modestly on the full test sets, while on the synthetic Bouc-Wen tasks, it improves the in-range multisine setting but substantially deteriorates performance in the transfer setting, where the same underlying system is driven by an unseen forcing process at test time. This is consistent with the view that the linear latent rollout provides a stronger bias toward generalization under shifted exogenous inputs, whereas the nonlinear variant increases modeling flexibility but is more prone to fitting forcing-specific patterns.

This pattern is clearest on shifted-input subsets. As shown in Figure 1b and the appendix tables, both ablations degrade on out-of-range (OOR) real-world samples. On EPF-DE OOR samples, the MLP encoder drops more sharply, while the nonlinear-update ablation also underperforms the base model. Favorita OOR samples show SLiDE outperforming both ablations by 18% in RMSLE, while in-range performance is mostly unaffected. Together, these results indicate that both the structured encoder and shared linear latent rollout contribute to SLiDE's stronger generalization when exogenous inputs move beyond the training regime.

We further examine the model design through additional input-map and encoder ablations in Appendix D.1 and block-structure ablations in Appendix D.2. These experiments show that the proposed design remains competitive with more flexible alternatives and characterize its sensitivity to the chosen number of blocks. Finally, Appendix D.3 evaluates robustness to imperfect exogenous inputs, showing that SLiDE remains competitive under both missing and noisy future inputs.

7 Conclusion and Limitations

In this work, we introduced SLiDE, a structured architecture for exogenous multi-step forecasting inspired by Koopman-style latent linear modeling. SLiDE combines a structured lag-invariant recurrent encoder for reconstructing the forecast-start latent state with a shared linear latent rollout that propagates this state under additive exogenous forcing. Empirically, SLiDE achieves state-of-the-

art performance across real-world energy and retail benchmarks while remaining computationally efficient. Its advantage is particularly clear when future exogenous inputs shift beyond the training regime, where it outperforms the considered baselines on both real-world and synthetic tasks. Overall, our results suggest that carefully structured latent-state estimation together with a shared linear latent rollout is an effective design principle for forecasting with exogenous inputs.

Limitations. Our focus is on forecasting accuracy together with computational and memory efficiency. SLiDE inherits the strengths but also the modeling assumptions of Koopman-style latent representations. Its suitability may depend on how well a forecasting problem can be described by an approximately linear latent dynamics model. This could be more restrictive for systems with richer latent structure, for example, those not well captured by low-dimensional Koopman approximations or involving continuous spectra, which we do not study systematically here. As exogenous inputs directly influence the latent rollout, missing, noisy, or misaligned inputs affect forecast quality. Our experiments with missing or noisy inputs (see Appendix D.3) suggest that SLiDE is relatively robust compared to the considered baselines. Moreover, we only evaluate a limited set of shifted-input scenarios, namely out-of-range covariates with a limited number of samples per dataset and a synthetic transfer setting. Extending the analysis to a wider range of covariate shifts and exploring more adaptive latent operators, for example, context-dependent variants as in Li et al. [2025], are valuable directions for future work.

Acknowledgments and Disclosure of Funding

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), [reference number 260250]. Cette recherche a été partiellement financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence 260250]. Ce projet de recherche no 324302 est rendu possible grâce au financement du Fonds de recherche du Québec. The authors gratefully acknowledge funding from the Helmholtz Association under grant no. VH-NG-1727 and the Helmholtz Association's Initiative and Networking Fund through Helmholtz AI and the HAICORE@KIT partition. The authors gratefully acknowledge the computing time provided on the high-performance computer HoreKa by the National High-Performance Computing Center at KIT (NHR@KIT). This center is jointly supported by the Federal Ministry of Research, Technology and Space and the Ministry of Science, Research and the Arts of Baden-Württemberg, as part of the National High-Performance Computing (NHR) joint funding program. HoreKa is partly funded by the German Research Foundation (DFG).

During the preparation of this work, the authors used AI-assisted tools for language editing and code development. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

Omri Azencot, N. Benjamin Erichson, Vanessa Lin, and Michael Mahoney. Forecasting Sequential Data Using Consistent Koopman Autoencoders. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 475–485. PMLR, 2020.

Gerben I. Beintema, Maarten Schoukens, and Roland Tóth. Continuous-time identification of dynamic state-space models by deep subspace encoding. In *International Conference on Learning Representations*, 2023.

George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 5th edition, 2015.

Steven L. Brunton, Marko Budišić, Eurika Kaiser, and J. Nathan Kutz. Modern Koopman Theory for Dynamical Systems. *SIAM Review*, 64(2):229–340, 2022. doi: 10.1137/21M1401243.

Si-An Chen, Chun-Liang Li, Sercan Ö. Arık, Nathanael Christian Yoder, and Tomas Pfister. TSMixer: An All-MLP Architecture for Time Series Forecasting. *Transactions on Machine Learning Research*, 2023.

Alexis Cook, DanB, inversion, and Ryan Holbrook. Store Sales - Time Series Forecasting. `https://kaggle.com/competitions/store-sales-time-series-forecasting`, 2021. Kaggle.

Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan K. Mathur, Rajat Sen, and Rose Yu. Long-term Forecasting with TiDE: Time-series Dense Encoder. *Transactions on Machine Learning Research*, 2023.

Hadi Fanaee-T. Bike Sharing. UCI Machine Learning Repository (Dataset), 2013. doi: 10.24432/C5W894.

Robert Fildes, Shaohui Ma, and Stephan Kolassa. Retail forecasting: Research and practice. *International Journal of Forecasting*, 38(4):1283–1318, 2022. doi: 10.1016/j.ijforecast.2019.06.004.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *First Conference on Language Modeling*, 2024.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022.

Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific Data*, 6(1):96, 2019. doi: 10.1038/s41597-019-0103-9.

Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan Kościsz, Dennis Bader, Frédérick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and Gaël Grosch. Darts: User-Friendly Modern Machine Learning for Time Series. *Journal of Machine Learning Research*, 23(124):1–6, 2022.

Tao Hong, Pierre Pinson, Yi Wang, Rafał Weron, Dazhi Yang, and Hamidreza Zareipour. Energy Forecasting: A Review and Outlook. *IEEE Open Access Journal of Power and Energy*, 7:376–388, 2020. doi: 10.1109/OAJPE.2020.3029979.

Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. doi: 10.18637/jss.v027.i03.

Lucian Cristian Iacob, Gerben Izaak Beintema, Maarten Schoukens, and Roland Tóth. Deep Identification of Nonlinear Systems in Koopman Form. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2288–2293, 2021. doi: 10.1109/CDC45484.2021.9682946.

Lucian Cristian Iacob, Roland Tóth, and Maarten Schoukens. Koopman form of nonlinear systems with inputs. *Automatica*, 162:111525, 2024. doi: 10.1016/j.automatica.2024.111525.

Lucian Cristian Iacob, Máté Szécsi, Gerben Izaak Beintema, Maarten Schoukens, and Roland Tóth. Learning Koopman Models From Data Under General Noise Conditions. arXiv preprint: arXiv:2507.09646, 2025.

Eurika Kaiser, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of Koopman eigenfunctions for control. *Machine Learning: Science and Technology*, 2(3):035023, 2021. doi: 10.1088/2632-2153/abf0f5.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift. In *International Conference on Learning Representations*, 2022.

B. O. Koopman. Hamiltonian Systems and Transformation in Hilbert Space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931. doi: 10.1073/pnas.17.5.315.

Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018. doi: 10.1016/j.automatica.2018.03.046.

Jesus Lago, Grzegorz Marcjasz, Bart De Schutter, and Rafał Weron. Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. *Applied Energy*, 293:116983, 2021. doi: 10.1016/j.apenergy.2021.116983.

Henning Lange, Steven L. Brunton, and J. Nathan Kutz. From Fourier to Koopman: Spectral Methods for Long-term Time Series Prediction. *Journal of Machine Learning Research*, 22(41):1–38, 2021.

Zhaoyang Li, Minghao Han, and Xunyuan Yin. MamKO: Mamba-based Koopman operator for modeling and predictive control. In *International Conference on Learning Representations*, 2025.

Zhengyu Li, Xiangfei Qiu, Yuhan Zhu, Xingjian Wu, Jilin Hu, Chenjuan Guo, and Bin Yang. GCGNet: Graph-Consistent Generative Network for Time Series Forecasting with Exogenous Variables. In *International Conference on Learning Representations*, 2026.

Bryan Lim, Sercan Ö. Arık, Nicolas Loeff, and Tomas Pfister. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021. doi: 10.1016/j.ijforecast.2021.03.012.

Yong Liu, Chenyu Li, Jianmin Wang, and Mingsheng Long. Koopa: Learning Non-stationary Time Series Dynamics with Koopman Predictors. *Advances in Neural Information Processing Systems*, 36:12271–12290, 2023. doi: 10.52202/075280-0538.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018. doi: 10.1038/s41467-018-07210-0.

Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. VAMPnets for deep learning of molecular kinetics. *Nature Communications*, 9(1):5, 2018. doi: 10.1038/s41467-017-02388-1.

Jean-Philippe Noël and Maarten Schoukens. Hysteretic Benchmark with a Dynamic Nonlinearity. 4TU.ResearchData, Version 1 (Dataset), 2020. doi: 10.4121/12967592.v1.

Kin G. Olivares, Cristian Challú, Azul Garza, Max Mergenthaler Canseco, and Artur Dubrawski. NeuralForecast: User friendly state-of-the-art neural forecasting models. PyCon Salt Lake City, Utah, US 2022, 2022. URL `https://github.com/Nixtla/neuralforecast`.

Kin G. Olivares, Cristian Challu, Grzegorz Marcjasz, Rafał Weron, and Artur Dubrawski. Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx. *International Journal of Forecasting*, 39(2):884–900, 2023. doi: 10.1016/j.ijforecast.2022.03.001.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020.

Antonio Orvieto, Samuel L. Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting Recurrent Neural Networks for Long Sequences. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 26670–26698. PMLR, 2023.

Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016. doi: 10.1137/15M1013857.

Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Generalizing Koopman Theory to Allow for Inputs and Control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018. doi: 10.1137/16M1062296.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020. doi: 10.1016/j.ijforecast.2019.07.001.

M. Schoukens and J. P. Noël. Three Benchmarks Addressing Open Challenges in Nonlinear System Identification. *IFAC-PapersOnLine*, 50(1):446–451, 2017. doi: 10.1016/j.ifacol.2017.08.071.

Nicholas S. Selby and H. Harry Asada. Learning of Causal Observable Functions for Koopman-DFL Lifting Linearization of Nonlinear Controlled Systems and Its Application to Excavation Automation. *IEEE Robotics and Automation Letters*, 6(4):6297–6304, 2021. doi: 10.1109/LRA.2021.3092256.

Oleksandr Shchur, Abdul Fatir Ansari, Caner Turkmen, Lorenzo Stella, Nick Erickson, Pablo Guerron, Michael Bohlke-Schneider, and Yuyang Wang. Fev-bench: A Realistic Benchmark for Time Series Forecasting. arXiv preprint: arXiv:2509.26468, 2025.

James H. Stock and Mark W. Watson. Forecasting Output and Inflation: The Role of Asset Prices. *Journal of Economic Literature*, 41(3):788–829, 2003. doi: 10.1257/002205103322436197.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning Koopman Invariant Subspaces for Dynamic Mode Decomposition. In *Advances in Neural Information Processing Systems*, volume 30, pages 1130–1140, 2017.

Julius Trebbien, Leonardo Rydin Gorjão, Aaron Praktiknjo, Benjamin Schäfer, and Dirk Witthaut. Understanding electricity prices beyond the merit order principle using explainable AI. *Energy and AI*, 13:100250, 2023. doi: 10.1016/j.egyai.2023.100250.

Eleni I. Vlahogianni, Matthew G. Karlaftis, and John C. Golias. Short-term traffic forecasting: Where we are and where we're going. *Transportation Research Part C: Emerging Technologies*, 43:3–19, 2014. doi: 10.1016/j.trc.2014.01.005.

Rui Wang, Yihe Dong, Sercan Ö. Arık, and Rose Yu. Koopman Neural Operator Forecaster for Time-series with Temporal Distributional Shifts. In *International Conference on Learning Representations*, 2023.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yun-Zhong Qiu, Jianmin Wang, and Mingsheng Long. TimeXer: Empowering Transformers for Time Series Forecasting with Exogenous Variables. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0015.

Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of Chemical Physics*, 148(24):241703, 2018. doi: 10.1063/1.5011399.

Rafał Weron. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, 30(4):1030–1081, 2014. doi: 10.1016/j.ijforecast.2014.08.008.

Mitchell Wortsman, Maxwell C. Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning Neural Network Subspaces. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 11217–11227. PMLR, 2021.

Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning Deep Neural Network Representations for Koopman Operators of Nonlinear Dynamical Systems. In *2019 American Control Conference (ACC)*, pages 4832–4839, 2019. doi: 10.23919/ACC.2019.8815339.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11121–11128, 2023. doi: 10.1609/aaai.v37i9.26317.

Yitian Zhang, Liheng Ma, Antonios Valkanas, Boris N. Oreshkin, and Mark Coates. SKOLR: Structured Koopman Operator Linear RNN for Time-Series Forecasting. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 75734–75756. PMLR, 2025.

A Derivation of the reconstructability map

In the following section, we provide a condensed derivation of the reconstructability map from Iacob et al. [2025] in the context of our method. We write $\boldsymbol{z}_{\tau+1} = A\boldsymbol{z}_\tau + \mathcal{B}\boldsymbol{u}_{\tau+1} := \mathcal{F}(\boldsymbol{z}_\tau, \boldsymbol{u}_{\tau+1})$ and use $\circ_j\mathcal{F}$ for j successive applications of $\mathcal{F}$. As in the main text, $\boldsymbol{y}_{\tau,[m]} = [\boldsymbol{y}_{\tau+1}^\top, \dots, \boldsymbol{y}_{\tau+m}^\top]^\top$ and $\boldsymbol{u}_{\tau,[m]} = [\boldsymbol{u}_{\tau+1}^\top, \dots, \boldsymbol{u}_{\tau+m}^\top]^\top$ (see the note on $\boldsymbol{u}$ below).

Observability map. Consider a window of m consecutive outputs $\boldsymbol{y}_{\tau,[m]}$, whose first sample is $\boldsymbol{y}_{\tau+1} = C\boldsymbol{z}_{\tau+1}$. Propagating $\boldsymbol{z}_{\tau+1}$ through the inputs that follow gives

$$\boldsymbol{y}_{\tau+1+j} = C\,(\circ_j\mathcal{F})(\boldsymbol{z}_{\tau+1}, \boldsymbol{u}_{\tau+1,[j]}), \qquad j = 0, \dots, m-1, \tag{14}$$

where, for $j = 0$, $\boldsymbol{u}_{\tau+1,[0]}$ is empty and $\circ_0\mathcal{F}$ is the identity, so that $\boldsymbol{y}_{\tau+1} = C\boldsymbol{z}_{\tau+1}$. Stacking these m outputs gives

$$\boldsymbol{y}_{\tau,[m]} = \Gamma_m(\boldsymbol{z}_{\tau+1}, \boldsymbol{u}_{\tau+1,[m-1]}). \tag{15}$$

Adapting Iacob et al. [2025] to our indexing, local observability from m output samples requires

$$\underbrace{\begin{bmatrix}\Gamma_m(\boldsymbol{z}_{\tau+1}, \boldsymbol{u}_{\tau+1,[m-1]})\\ \Upsilon(\boldsymbol{z}_{\tau+1})\end{bmatrix}}_{\mathcal{O}_{\boldsymbol{z},m}(\boldsymbol{z}_{\tau+1}, \boldsymbol{u}_{\tau+1,[m-1]})} = \begin{bmatrix}\boldsymbol{y}_{\tau,[m]}\\ 0\end{bmatrix}, \tag{16}$$

where $\Upsilon(\cdot)$ are nonlinear constraints on the lifted state that ensure the linear dynamics A are equivalent to the original nonlinear system. Inverting $\mathcal{O}_{\boldsymbol{z},m}$ yields the observability map

$$\boldsymbol{z}_{\tau+1} = \Lambda_{\boldsymbol{z},m}(\boldsymbol{y}_{\tau,[m]}, \boldsymbol{u}_{\tau+1,[m-1]}), \tag{17}$$

with $\Lambda_{\boldsymbol{z},m} : \mathbb{R}^{mn_y} \times \mathbb{R}^{(m-1)n_u} \to \mathbb{R}^{n_z}$.

Reconstructability map. Since we prefer to determine the current state from *past* data, we propagate the reconstructed window-start state to the end of the window, which takes $m-1$ forward steps:

$$\begin{aligned}\boldsymbol{z}_{\tau+m} &= (\circ_{m-1}\mathcal{F})(\boldsymbol{z}_{\tau+1}, \boldsymbol{u}_{\tau+1,[m-1]})\\ &= (\circ_{m-1}\mathcal{F})\big(\Lambda_{\boldsymbol{z},m}(\boldsymbol{y}_{\tau,[m]}, \boldsymbol{u}_{\tau+1,[m-1]}),\, \boldsymbol{u}_{\tau+1,[m-1]}\big)\\ &=: \Pi_{\boldsymbol{z},m}(\boldsymbol{y}_{\tau,[m]}, \boldsymbol{u}_{\tau+1,[m-1]}),\end{aligned} \tag{18}$$

where $\Pi_{\boldsymbol{z},m} : \mathbb{R}^{mn_y} \times \mathbb{R}^{(m-1)n_u} \to \mathbb{R}^{n_z}$ is the reconstructability map. Reconstruction requires a sufficiently large window m.

Full-history encoder. To reconstruct $\boldsymbol{z}_k$ at present time k from the full past trajectory $\boldsymbol{y}_{[k]} = \boldsymbol{y}_{0,[k]}$, we take $\tau = 0$ and $m = k$:

$$\boldsymbol{z}_k = \Pi_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{1,[k-1]}). \tag{19}$$

This map does not depend on $\boldsymbol{u}_1$, the input aligned with the first observation. Our parameterized encoder nonetheless takes the full aligned sequence $\boldsymbol{u}_{[k]}$ so that the same input–output structure is used at every time step:

$$\hat{\boldsymbol{z}}_k = \Pi^{\psi}_{\boldsymbol{z},k}(\boldsymbol{y}_{[k]}, \boldsymbol{u}_{[k]}), \tag{20}$$

where $\boldsymbol{u}_1$ is an additional input. Computing $\Pi_{\boldsymbol{z},k}$ analytically is infeasible because it requires inverting $\mathcal{O}_{\boldsymbol{z},k}$ to obtain $\Lambda_{\boldsymbol{z},k}$, so $\Pi^{\psi}_{\boldsymbol{z},k}$ has learnable parameters $\psi \in \Psi \subseteq \mathbb{R}^{n_\psi}$.

B Additional methodology details

B.1 Characterizing the departure from the exact Koopman form

The exact Koopman form has a state- and input-dependent input map

$$\boldsymbol{z}_{\tau+1} = A\boldsymbol{z}_\tau + B(\boldsymbol{z}_\tau, \boldsymbol{u}_{\tau+1})\boldsymbol{u}_{\tau+1}. \tag{21}$$

However, as detailed in Iacob et al. [2025], it may be desirable to trade accuracy for simplicity of the model, e.g. by choosing a suitable form of the input function $\mathcal{B}$ in Equation (7a). In our case, we do not only choose the input map to be state-independent, but instead of explicitly parameterizing the matrix-valued map $\mathcal{B}$, our implementation directly parameterizes the resulting whole input forcing term as $\mathrm{MLP}_{\boldsymbol{u}}(\boldsymbol{u}) \in \mathbb{R}^{n_z}$. The following characterizes what this input-decoupled model retains relative to the exact form.

The input-decoupled form is exact if there exists a finite-dimensional lifting Φ for which the effect of the input on the lifted dynamics is independent of the lifted state

$$\Phi(f(\boldsymbol{x}, \boldsymbol{u})) = A\Phi(\boldsymbol{x}) + g(\boldsymbol{u}). \tag{22}$$

This structure is analogous to a Hammerstein model in the lifted coordinates, where a static nonlinear transformation of the input drives otherwise linear latent dynamics. If, instead, the exact lifted input forcing depends on both $\boldsymbol{z}$ and $\boldsymbol{u}$, as permitted by the general Koopman form $B(\boldsymbol{z}, \boldsymbol{u})\boldsymbol{u}$, the state-independent forcing $g(\boldsymbol{u})$ constitutes an approximation.

When the input-decoupled form is not exact, we can characterize the error induced specifically by removing the latent-state dependence from the input forcing. Let $B(\boldsymbol{z}, \boldsymbol{u})\boldsymbol{u}$ denote the exact lifted forcing and let $g(\boldsymbol{u})$ denote its state-independent approximation. We assume that their mismatch is uniformly bounded over the relevant domain $\mathcal{D}$ of lifted states and inputs:

$$\sup_{(\boldsymbol{z},\boldsymbol{u})\in\mathcal{D}} \|B(\boldsymbol{z}, \boldsymbol{u})\boldsymbol{u} - g(\boldsymbol{u})\|_2 \leq \epsilon. \tag{23}$$

Furthermore, assume that the linear latent operator A is Schur stable, i.e.,

$$\rho(A) < 1. \tag{24}$$

For any λ satisfying $\rho(A) < \lambda < 1$, there then exists a constant $c \geq 1$ such that

$$\left\|A^h\right\|_2 \leq c\lambda^h, \qquad h \geq 0. \tag{25}$$

To isolate the error introduced by the state-independent forcing, consider the exact and input-decoupled lifted dynamics under the same input sequence:

$$\boldsymbol{z}_{h+1} = A\boldsymbol{z}_h + B(\boldsymbol{z}_h, \boldsymbol{u}_{h+1})\boldsymbol{u}_{h+1}, \tag{26a}$$
$$\tilde{\boldsymbol{z}}_{h+1} = A\tilde{\boldsymbol{z}}_h + g(\boldsymbol{u}_{h+1}), \tag{26b}$$

with identical initial lifted states,

$$\boldsymbol{z}_0 = \tilde{\boldsymbol{z}}_0. \tag{27}$$

Define the trajectory discrepancy and the corresponding one-step forcing mismatch as

$$\mathbf{e}_h := \boldsymbol{z}_h - \tilde{\boldsymbol{z}}_h, \tag{28a}$$
$$\boldsymbol{\delta}_h := B(\boldsymbol{z}_h, \boldsymbol{u}_{h+1})\boldsymbol{u}_{h+1} - g(\boldsymbol{u}_{h+1}). \tag{28b}$$

By assumption,

$$\|\boldsymbol{\delta}_h\|_2 \leq \epsilon \tag{29}$$

for all considered state-input pairs. Subtracting the two latent dynamics gives

$$\mathbf{e}_{h+1} = A\mathbf{e}_h + \boldsymbol{\delta}_h, \qquad \mathbf{e}_0 = \mathbf{0}. \tag{30}$$

Unrolling this recursion yields

$$\mathbf{e}_h = \sum_{j=0}^{h-1} A^{h-1-j}\boldsymbol{\delta}_j. \tag{31}$$

Therefore,

$$\|\mathbf{e}_h\|_2 \leq \sum_{j=0}^{h-1} \left\|A^{h-1-j}\right\|_2 \|\boldsymbol{\delta}_j\|_2 \tag{32}$$

$$\leq c\epsilon \sum_{j=0}^{h-1} \lambda^{h-1-j} \tag{33}$$

$$= c\epsilon \sum_{j=0}^{h-1} \lambda^{j} \tag{34}$$

$$= c\frac{1-\lambda^h}{1-\lambda}\epsilon. \tag{35}$$

Consequently,

$$\limsup_{h\to\infty} \|\mathbf{e}_h\|_2 \leq \frac{c}{1-\lambda}\epsilon. \tag{36}$$

Thus, under Schur-stable latent dynamics, a uniformly bounded error resulting from the state-independent approximation of the input forcing does not accumulate without bound with the forecast horizon.

This bound isolates the approximation introduced by replacing the exact forcing $B(\boldsymbol{z}, \boldsymbol{u})\boldsymbol{u}$ with $g(\boldsymbol{u})$ while keeping the same latent operator A. Therefore, it does not constitute a bound on the total error of the learned forecasting model. In our implementation, the latent operator is learned as $A_\theta = W$ and the forcing is directly parameterized as

$$g_\theta(\boldsymbol{u}) := \mathrm{MLP}_{\boldsymbol{u}}(\boldsymbol{u}). \tag{37}$$

The exact Koopman operator A is unknown in our setting. However, our spectral analysis in Appendix E shows empirically that the learned W is Schur stable across the considered tasks. Appendix D.1.1 further provides a specific ablation experiment, replacing SLiDE's input map with a state-dependent alternative that stays closer to the structure proposed by the exact model form.

B.2 Patching

For time series with a long history k and forecast horizon H, we may optionally pre-process the target time series by patching together consecutive observations. We define a patch length P to transform the target time series to have $k' = k/P$ past values and $H' = H/P$ future values. For $\boldsymbol{y}_\tau^{\text{patch}} \in \mathbb{R}^{P\cdot n_y}$ and $\tau \in \{1, \ldots, k' + H'\}$, we define:

$$\boldsymbol{y}_\tau^{\text{patch}} \triangleq \left[\boldsymbol{y}_{P(\tau-1)+1}, \ldots, \boldsymbol{y}_{P\tau}\right]^\top. \tag{38}$$

Within each patch, the consecutive time steps and channels of $\boldsymbol{y}$ are flattened into a space $\mathbb{R}^{P\cdot n_y}$. This removes the sequential relationship of consecutive observations within the patch, treating it instead as simultaneous observations of a higher-dimensional variable. The range of defined time indices is reduced by a factor of P. This same patching mechanism is then analogously applied for the input sequence $\boldsymbol{u}$.

We begin by reframing our target time series from $\boldsymbol{y} \in \mathbb{R}^{n_y}$ to $\boldsymbol{y}^{\text{patch}} \in \mathbb{R}^{P\cdot n_y}$ following the patching process described in (38). The transformed target variable is equivalent to some other target $\boldsymbol{y}'$, with a different forecast horizon $H' = H/P$ and dimension $n'_y = P \cdot n_y$. Our model treats $\boldsymbol{y}^{\text{patch}}$ as the variable of interest for forecasting, and we can subsequently obtain $\boldsymbol{y}$ by inverting the patching process.

C Additional details on "OOR" results

For the real-world datasets, *out-of-range exogenous inputs* provide a concrete instance of *shifted exogenous inputs*. These are test samples whose future-known exogenous values fall outside the training input regime. This setting does not imply a general distribution shift, but isolates a practical form of extrapolation in the future input process.

Table 2: Number of samples, here denoted as out-of-range (OOR) samples, for which an exogenous input value exceeds the training range per EPF dataset.

Dataset	#OOR samples
NP	18
PJM	0
EPEX BE	16
EPEX FR	16
EPEX DE	93

Table 3: Detailed EPF DE OOR Results

Model	OOR		In-Range		overall	
	MAE	MSE	MAE	MSE	MAE	MSE
SLiDE	**0.305**	**0.278**	**0.254**	**0.156**	**0.265**	**0.182**
TimeXer	0.348	0.384	0.259	0.166	0.278	0.213
GCGNet	0.349	0.356	0.267	0.166	0.285	0.206
TSMixer	0.367	0.418	0.261	0.165	0.283	0.218
TiDE	0.379	0.400	0.272	0.172	0.295	0.220
Koop.-Subnet	0.392	0.444	0.291	0.206	0.312	0.257
DLinear	0.399	0.471	0.282	0.188	0.307	0.248
CT-Subnet	0.407	0.452	0.294	0.192	0.318	0.247
MamKO	0.415	0.495	0.301	0.217	0.325	0.276
SKOLR	0.502	0.698	0.385	0.360	0.410	0.432
Base	**0.305**	**0.278**	**0.254**	**0.156**	**0.265**	**0.182**
Nonlinearity	0.311	0.285	0.258	0.160	0.269	0.186
MLP Encoder	0.348	0.342	0.263	0.165	0.281	0.203

EPF datasets. Table 2 reports the number of such OOR samples for each dataset in the EPF benchmark. The phenomenon is relatively rare in most markets, but occurs most frequently in EPF-DE, which contains 93 OOR test days, substantially more than the other EPF datasets. All EPF datasets contain 437 test days in total. We therefore focus our detailed analysis on EPF-DE, as it provides the clearest real-world setting for studying generalization beyond the training support of future-known covariates.

Table 3 compares model performance on the OOR subset, the complementary in-range subset, and the full EPF-DE test set. SLiDE performs best on the OOR subset, achieving the lowest MAE and MSE among all models. The margin over the strongest forecasting baselines is substantially larger on OOR samples than on the in-range subset, indicating that SLiDE's advantage is concentrated in cases where future exogenous inputs exceed the range seen during training. The same trend is reflected in the ablations: both the nonlinear-update and MLP-encoder variants degrade on the OOR subset, with the MLP encoder showing the larger drop. This supports the view that both the structured encoder and the shared linear latent rollout contribute to the improved behavior under shifted input conditions.

Favorita. We also perform an analogous analysis on the Favorita dataset. Here, we define a test sample as OOR if the number of promoted items for the corresponding product group, evaluated over the forecast horizon, exceeds the range observed during training for this particular group. Under this definition, 60 out of 1,579 test samples qualify as OOR. Although these samples make up only a small fraction of the test set, Table 4 shows that this is precisely the region in which SLiDE separates most clearly from the baselines: it achieves the best RMSLE on the OOR subset, while differences on the in-range subset are much smaller. The ablation results follow the same pattern, with both the MLP encoder and nonlinear latent update degrading performance on the OOR subset. Figure 4 displays an example forecast for a test sample with out-of-range promotion inputs, where SLiDE stays closer to the ground truth than either the MLP-encoder or nonlinear-rollout ablation.

Table 4: Detailed Favorita OOR Results

Model	OOR RMSLE	In-Range RMSLE	overall RMSLE
SLiDE	**0.670**	0.406	**0.419**
CT-Subnet	0.752	0.410	0.428
TiDE	0.761	0.405	0.424
TSMixer	0.805	**0.400**	0.422
Koop.-Subnet	0.808	0.417	0.438
TimeXer	1.116	0.415	0.461
MamKO	1.179	0.440	0.489
DLinear	1.237	0.427	0.483
GCGNet	1.252	0.424	0.482
SKOLR	1.288	0.415	0.478
Base	**0.670**	0.406	**0.419**
MLP Encoder	0.819	0.401	0.424
Nonlinearity	0.877	**0.400**	0.428

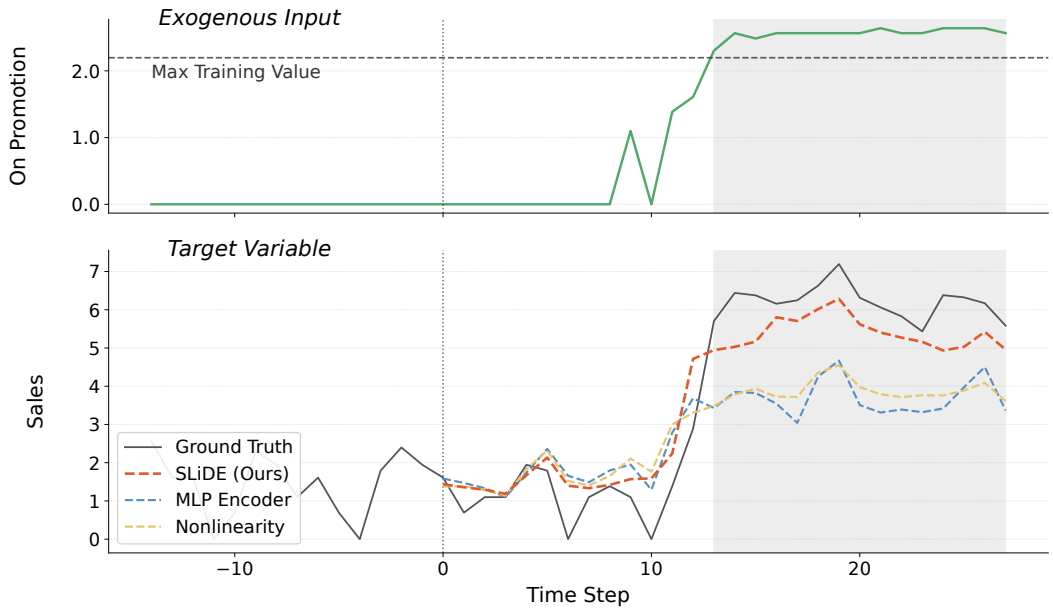


Figure 4: Favorita example with out-of-range promotion inputs. The top panel shows the future-known exogenous input, and the bottom panel the corresponding sales forecasts. Once the promotion signal exceeds the maximum training value, SLiDE stays closer to the ground truth than either the MLP-encoder or nonlinear-rollout ablation.

Although this analysis is based on a relatively small number of samples, it is consistent with the broader observation that SLiDE is particularly effective when future exogenous inputs move beyond the support observed during training.

Bouc-Wen Transfer. Figure 5 provides an illustrative Bouc-Wen transfer trajectory. In this example, SLiDE remains closer to the ground truth than strong forecasting baselines across much of the rollout, especially in intervals with out-of-range input values. Figure 6 shows an example trajectory comparing SLiDE to the main ablations. The nonlinear latent update produces substantially larger amplitude errors, while the MLP encoder also degrades performance relative to the base model.

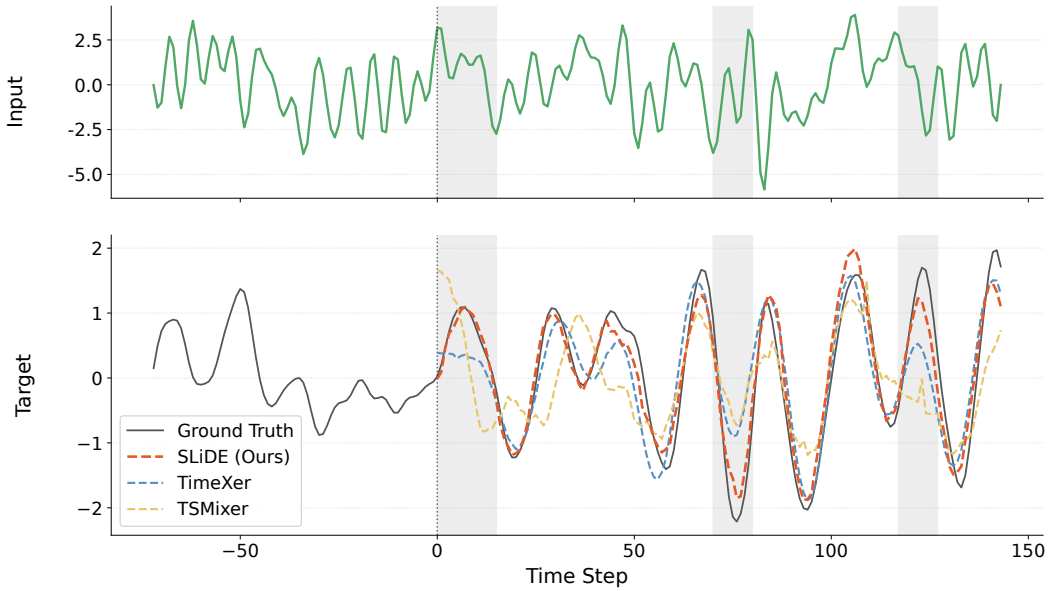


Figure 5: Illustrative Bouc-Wen transfer trajectory for SLiDE and two baselines. The top panel shows the exogenous input process. The bottom panel compares the ground-truth target trajectory with forecasts from SLiDE, TimeXer and TSMixer. The shaded regions highlight intervals in which SLiDE tracks the ground-truth noticeably better than both of the baselines shown here.

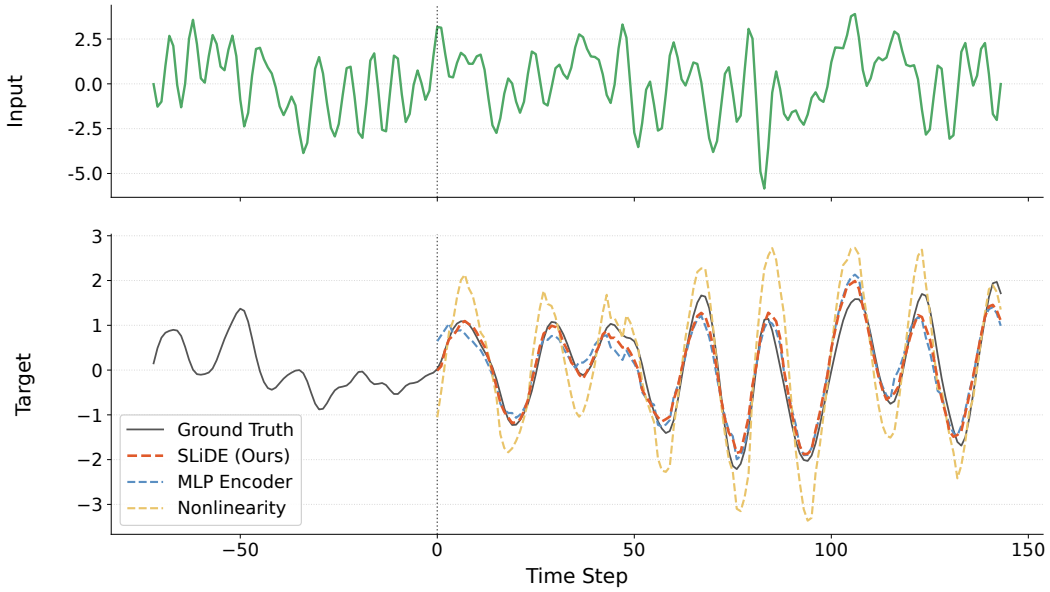


Figure 6: Illustrative Bouc-Wen transfer trajectory for SLiDE and the two main ablations. The top panel shows the exogenous input process. The bottom panel compares the ground-truth target trajectory with forecasts from SLiDE, the MLP-encoder ablation, and the nonlinear-rollout ablation. While the MLP-encoder ablation remains relatively close to the base model, the nonlinear latent update produces substantially larger amplitude errors.

D Additional ablations

D.1 Input-map and encoder ablations

D.1.1 State-dependent input map

To assess the effect of removing the latent-state dependence from the exogenous forcing, we implement a state-dependent input map following Koopman-Subnet. An MLP takes the current latent state and input as arguments and outputs a matrix $B(z, u) \in \mathbb{R}^{n_z \times n_u}$, such that the additive forcing is given by $B(z, u)u$.

Table 5: Ablation study of the state-dependent input map and the Transformer encoder, shown together with the nonlinear state-update and MLP-encoder ablations from Section 6. We highlight the **best** result for each metric.

Model	EPF Benchmark										Favorita	Bouc-Wen System			
	NP		PJM		EPEX BE		EPEX FR		EPEX DE		Sales	Multisine		Transfer	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	RMSLE	MAE	MSE	MAE	MSE
SLiDE	**0.210**	**0.167**	**0.157**	**0.064**	0.231	0.367	0.169	0.329	**0.265**	**0.182**	**0.419**	0.085	0.012	**0.296**	**0.139**
Nonlinearity	0.214	0.170	0.158	0.067	0.236	0.371	0.169	0.329	0.269	0.186	0.428	**0.035**	**0.002**	0.741	0.877
State-dependent map	0.218	0.174	0.160	0.068	0.234	0.360	0.174	0.328	0.277	0.207	0.427	0.064	0.007	0.370	0.312
MLP encoder	0.223	0.172	0.160	0.065	0.239	**0.352**	0.183	0.332	0.281	0.203	0.423	0.091	0.014	0.360	0.207
Attention-based encoder	0.214	0.173	0.159	0.069	**0.229**	0.354	**0.167**	**0.318**	0.267	0.184	0.426	0.075	0.010	0.297	0.140

The state-dependent map improves on the noiseless, fully deterministic Multisine task, as does the nonlinear rollout in Section 6, but it is worse on the real-world benchmarks and degrades sharply on the Transfer task, where the exogenous input process shifts at test time. The additional state dependence is therefore helpful only in that clean deterministic setting. The input-only forcing performs better on the real-world tasks and under shifted forcing, and it uses fewer parameters: the state-dependent MLP must output an $n_z \times n_u$ matrix rather than an n_z-dimensional forcing vector.

D.1.2 Attention-based encoder

We implement an attention-based encoder while keeping the linear rollout fixed. Tokens are embedded using the same MLPs for y_τ and u_τ and summed, then processed by a standard Transformer with RoPE self-attention. The final hidden state is used as $\hat{z}_k$. Among the ablations in Table 5, this encoder is the most competitive with SLiDE. However, this architecture suffers from an increase in parameter count and computational cost, since self-attention replaces the linear recurrent encoder, and does not provide a clear performance uplift relative to SLiDE.

D.2 Block-structure ablations

D.2.1 Selection of and sensitivity to block count

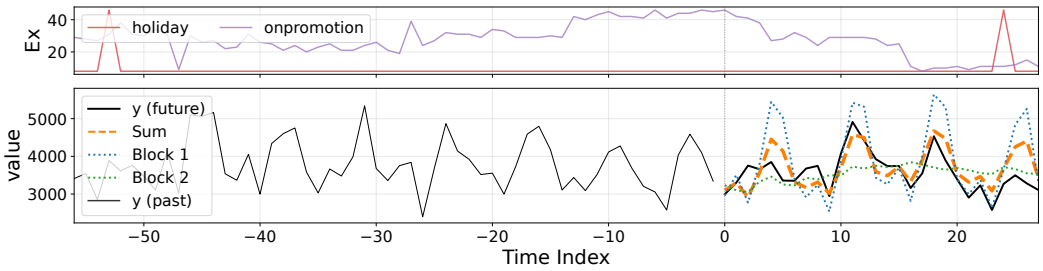


Figure 7: Forecast sample of SLiDE on the Favorita benchmark for predicting product sales. Top: The exogenous inputs, indicating the presence of holidays and number of related products on promotion. Bottom: Sales target and SLiDE forecast broken down into individual block forecasts. We observe that the two blocks learn different temporal patterns, which are combined to form the final forecast.

Figure 7 illustrates a Favorita forecast in which the two block contributions exhibit different temporal patterns and combine to form the final prediction. To further investigate and inform the choice of the number of blocks N, we ran a dedicated ablation with $N \in 1, 2, 4, 8$ in two regimes: (i) fixed per-block dimension, where the total parameter count grows with N, and (ii) fixed total latent dimension, where more blocks make W sparser and reduce parameters.

Table 6 reports the results. On average, performance improves from one to two or four blocks, with only minor differences between N=2 and N=4. Increasing the number of blocks to N=8 generally degrades performance, particularly in the fixed-total-dimension regime, where the average relative performance gap to the best-performing variant reaches 18.73%. We therefore selected N=2 as the default for its favorable performance–cost trade-off and excluded N from HPO to reduce computational cost.

Table 6: Block ablation results with fixed latent dimension (*Fixed per-block dimension*) and with latent dimension scaled inversely to the number of blocks (*Fixed total dimension*). We highlight the **first**, second and third results across both experiments for each metric.

	EPF Benchmark										Favorita	Bouc-Wen System			
Blocks	**NP**		**PJM**		**EPEX BE**		**EPEX FR**		**EPEX DE**		**Sales**	**Multisine**		**Transfer**	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	RMSLE	MAE	MSE	MAE	MSE
Fixed per-block dimension:															
1	0.214	0.173	0.159	0.065	0.235	0.380	**0.167**	0.344	0.269	0.191	0.423	**0.082**	**0.011**	0.306	0.148
2	0.210	0.167	0.157	0.064	0.231	0.367	0.169	0.329	0.265	0.182	**0.419**	0.085	0.012	0.296	0.139
4	0.209	**0.164**	0.157	0.064	0.231	0.366	0.169	0.324	0.265	0.182	0.424	0.086	0.012	0.306	0.149
8	0.211	0.169	**0.155**	**0.063**	**0.229**	**0.357**	0.172	**0.320**	0.264	0.184	0.428	0.091	0.014	0.327	0.169
Fixed total dimension:															
1	**0.208**	0.165	0.159	0.066	0.232	0.361	**0.167**	0.337	0.267	0.189	0.421	0.084	0.012	0.307	0.150
4	0.211	0.171	0.157	**0.063**	0.232	0.367	0.173	0.332	**0.262**	**0.180**	0.420	0.086	0.012	**0.291**	**0.134**
8	0.212	0.173	0.158	**0.063**	0.232	0.363	0.176	0.336	0.265	0.183	0.424	0.089	0.013	0.447	0.367

D.2.2 Explicit block diversity

While Figure 7 suggests that different blocks learn distinct temporal patterns, Figure 9 shows that their eigenvalue spectra largely overlap. To investigate whether explicitly enforcing diversity improves performance, we added a diversity loss that penalizes pairwise squared cosine similarity between block operators under the Frobenius inner product, following the cosine-similarity diversity regularization of Wortsman et al. [2021]. We evaluated weights w=1 and w=10 across $N \in \{1, 2, 4, 8\}$.

The effect on performance is negligible, with the mean relative performance change across dataset–metric pairs ranging from -0.36% to $+0.02\%$ across the tested configurations. Thus, explicitly encouraging block diversity does not yield systematic performance improvements in the considered experiments.

D.3 Robustness

D.3.1 Missing exogenous inputs

We evaluate robustness to missing exogenous inputs on EPF PJM and EPF DE, chosen to represent both a setting in which competing baselines are strong (PJM) and a setting in which SLiDE significantly outperforms (DE). For each time step, we independently drop all exogenous values with probability p according to a Bernoulli distribution. Missing values are replaced with zeros after dataset-wide standardization, and an additional binary indicator channel is appended to mark whether the exogenous values at that time step are observed or missing. We reuse the hyperparameter configuration selected in the main experiments and report average metrics over five independent runs with different random seeds. Figure 8 shows that SLiDE outperforms TimeXer and TSMixer at 30% and 60% missing values on EPF PJM, despite being slightly behind at full availability. SLiDE's performance degradation is comparable to the lighter TSMixer, while attention-based TimeXer degrades most sharply, demonstrating strong robustness relative to the most competitive baseline available.

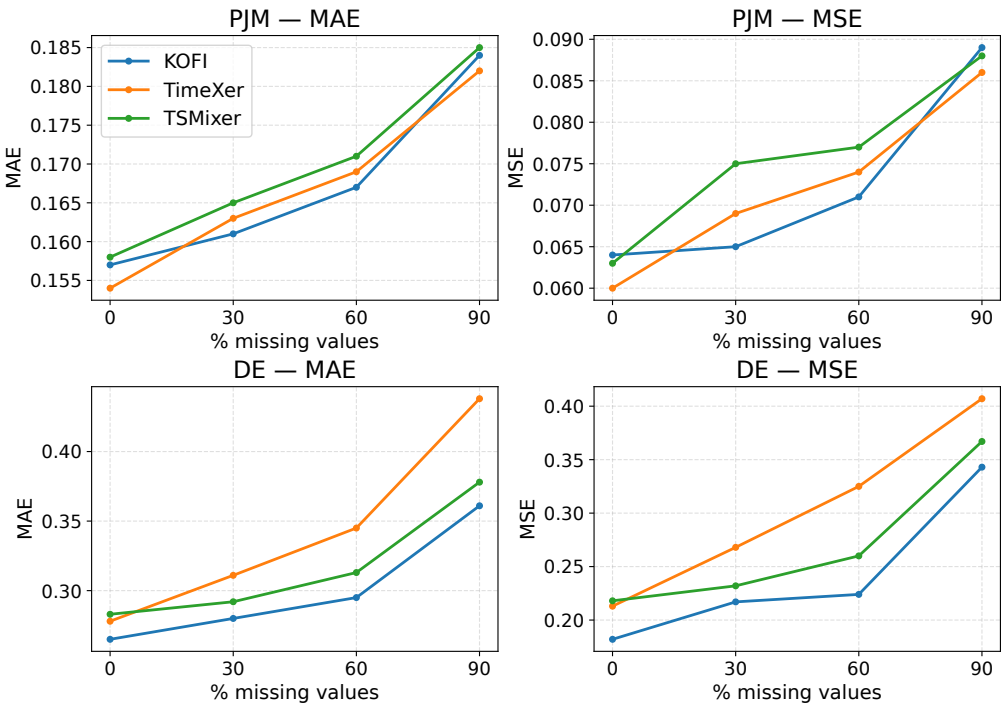


Figure 8: Evaluation metrics (MAE left, MSE right) of SLiDE, TimeXer and TSMixer against percentage of missing exogenous values for the EPF PJM (top) and DE (bottom) datasets. SLiDE outperforms baselines on PJM with 30% and 60% of missing values, and is significantly more robust than TimeXer, the second-best ranking model, on EPF DE.

D.3.2 Noisy future exogenous inputs

We note that our real-world benchmarks already contain errors in future exogenous inputs by construction: the future-known covariates in all five EPF datasets are themselves day-ahead forecasts, not realized values. The EPF results therefore already evaluate the models using future exogenous variables that contain realistic forecast uncertainty.

To additionally characterize robustness in a controlled way, we ran an input corruption experiment. We perturb only the future exogenous inputs in the horizon with Gaussian noise (standard deviation scaled to 50% of each input's standard deviation), keeping past values unperturbed. Each model is re-trained with its original hyperparameters on unperturbed past and noisy future inputs. We omit Favorita from this experiment since its future-known covariates (holidays, promotions, calendar features) are deterministically known in advance, so corrupting them would not correspond to a realistic uncertainty scenario.

SLiDE maintains its top average rank under this noisy regime. Notably, on Bouc–Wen Multisine, SLiDE now outperforms Koopman-Subnet and CT-Subnet, which outperform SLiDE on the task without noise addition but show particularly large relative degradation when the future forcing is corrupted. Results are reported in Table 7.

E Spectrum of the learned operator

We analyze the eigenvalue spectrum of the learned operator W across blocks and datasets, as shown in Figure 9. Although no explicit spectral constraint is imposed during training, all learned eigenvalues remain inside the unit circle for the considered models. Hence, the learned operators are Schur stable, empirically satisfying the stability assumption used in our error bound in App. B.1. Most eigenvalues lie well within the stability boundary, indicating predominantly fast-decaying latent modes. Note that W was initialized using PyTorch's default `nn.Linear` initialization, which, for sufficiently large

Table 7: Performance under noisy future exogenous inputs on the EPF benchmark and Bouc-Wen tasks. We highlight the **first**, second, and third results for each metric.

Model	EPF Benchmark										Bouc-Wen System			
	NP		PJM		EPEX BE		EPEX FR		EPEX DE		Multisine		Transfer	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
SLiDE	**0.219**	**0.181**	0.163	0.069	**0.236**	0.369	0.176	0.339	**0.278**	**0.203**	**0.243**	**0.096**	**0.336**	**0.178**
Forecasting Architectures:														
TimeXer	0.222	0.188	**0.158**	**0.064**	**0.236**	0.377	**0.173**	0.348	0.290	0.229	0.284	0.128	0.503	0.395
GCGNet	0.227	0.184	0.162	0.068	0.252	0.373	0.190	**0.315**	0.296	0.221	0.335	0.178	0.481	0.366
TSMixer	0.231	0.201	0.165	0.086	0.242	**0.343**	0.200	0.316	0.304	0.250	0.286	0.131	0.670	0.706
TiDE	0.247	0.206	0.164	0.075	0.255	0.357	0.201	0.350	0.309	0.247	0.281	0.126	0.494	0.383
Related System Identification or Control Baselines:														
Koop.-Subnet	0.265	0.234	0.195	0.097	0.259	0.372	0.198	0.336	0.334	0.289	0.250	0.100	0.463	0.337
CT-Subnet	0.257	0.208	0.185	0.089	0.263	0.396	0.210	0.386	0.337	0.288	0.262	0.109	0.768	0.885
MamKO	0.265	0.242	0.182	0.085	0.269	0.395	0.206	0.334	0.375	0.355	0.508	0.399	0.744	0.968

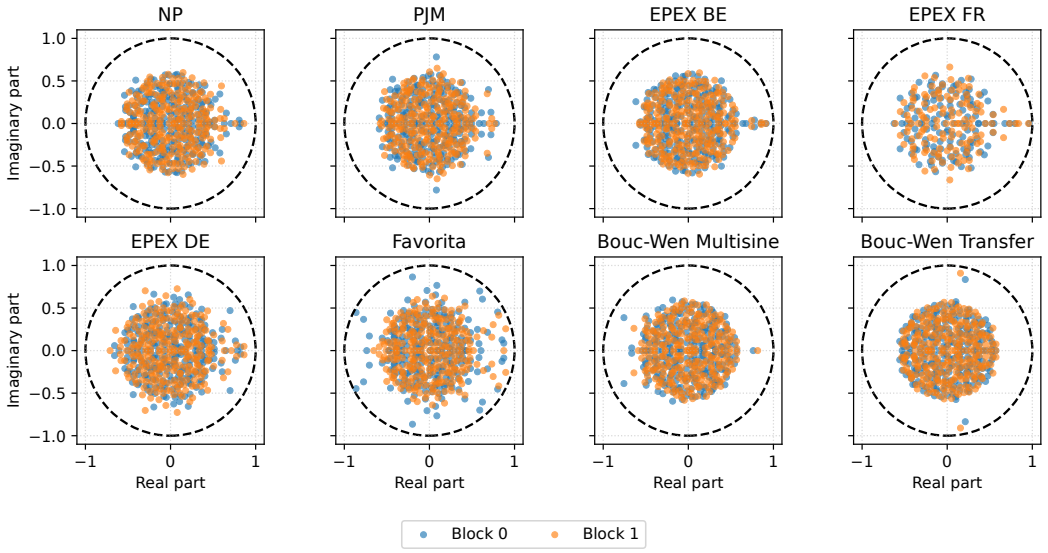


Figure 9: Eigenvalue spectra of the learned operator W across blocks and datasets. Each point denotes an eigenvalue of a block-specific operator and the dashed circle indicates the unit circle.

square matrices, yields an approximately uniform eigenvalue distribution within a disk of radius $1/\sqrt{3}$.

Across blocks, the eigenvalue distributions overlap substantially, with no clear partition into distinct frequency bands or decay timescales. The block structure therefore does not appear to induce a spectral decomposition of the dynamics. Instead, the results are consistent with the blocks providing a parameter-efficient collection of parallel latent dynamics, despite the corresponding block forecasts specializing to different temporal patterns (see Figure 7).

The Bouc–Wen benchmark provides an informative case study. Its linearized dynamics have a reported natural frequency of 35.59 Hz [Noël and Schoukens, 2020]. At a sampling rate of 750 Hz, this corresponds to discrete-time eigenvalue angles of approximately ± 2.38 rad ($\pm 137°$) for patch length $P = 8$ and ± 1.19 rad ($\pm 68°$) for $P = 4$. The dominant learned conjugate pairs roughly agree with these positions. This correspondence suggests that W captures meaningful oscillatory structure of the underlying system, while the remaining, generally faster-decaying modes provide additional flexibility for representing the nonlinear hysteretic dynamics.

F Experimental details

F.1 Additional information on datasets

Table 8: Summary of the datasets and covariate types used in our experiments.

Dataset	# Targets	# Series	Past	Future	Static	Context	Horizon	Loss	Metrics
EPF datasets	1	1	0	2	0	168	24	MAE	MAE, MSE
Favorita Sales	1	1,579	1	6	6	56	28	MSE (log)	RMSLE
Bouc-Wen tasks	1	1	0	1	0	288	144	MAE	MAE, MSE

Electricity Price Forecasting (EPF) Benchmark. The EPF benchmark [Lago et al., 2021] consists of five datasets of different electricity markets. For each dataset, the task is forecasting the hourly day-ahead electricity price for the next 24 hours, given two forecasted covariates for the next day. The two forecasted covariates differ for each dataset. All datasets include six years of hourly data.

Nord Pool (NP): The NP dataset contains price data of the European power market of the Nordic countries. The two additional covariates are the day-ahead load forecast and the day-ahead wind generation forecast.

Pennsylvania–New Jersey-Maryland (PJM): This dataset includes prices from the Commonwealth Edison (COMED) zone from the PJM market in the United States as the target time series. The covariates are two day-ahead load forecast series, a system load forecast and a zonal load forecast for the COMED zone.

European Power Exchange - Belgium (EPEX-BE): The EPEX-BE dataset contains the day-ahead prices for the Belgian bidding zone on the market operated by EPEX SPOT. The two covariates in this dataset are the day-ahead load forecast and the day-ahead generation forecast in France.

European Power Exchange - France (EPEX-FR): The EPEX-FR dataset contains the day-ahead prices for the French bidding zone on the market operated by EPEX SPOT. The two covariates in this dataset are the day-ahead load forecast and the day-ahead generation forecast.

European Power Exchange - Germany (EPEX-DE): The EPEX-DE dataset includes the day-ahead prices for the German bidding zone on the market operated by EPEX SPOT. The two covariates in this dataset are the day-ahead load forecast for the control area operated by the transmission system operator Amprion and the aggregated day-ahead wind and solar generation forecast in the zones operated by the transmission system operators Amprion, TenneT and 50Hertz.

Following Li et al. [2026], we use a 70%/10%/20% train/validation/test split, a forecast horizon of 24 hours and a context length of one week (168 hours). In contrast to their evaluation setup, however, we extract inference samples with a stride of 24 hours rather than 1 hour in order to better reflect the application setting and realistic future-covariate availability. We train all models using a Mean Absolute Error (MAE) loss $\mathcal{R} = \frac{1}{DH}\sum_{d=1}^{D}\sum_{h=1}^{H}\left|y_{k_d+h} - \hat{y}_{k_d,h}\right|$ and evaluate performance with MAE and Mean Squared Error (MSE) $\mathcal{R} = \frac{1}{DH}\sum_{d=1}^{D}\sum_{h=1}^{H}\left(y_{k_d+h} - \hat{y}_{k_d,h}\right)^2$. All datasets are standardized using mean and standard deviation statistics computed from the respective training sets, and performance is reported in standardized units.

Favorita Sales Dataset. The Favorita Sales Dataset [Cook et al., 2021] is a large-scale retail forecasting dataset that contains daily sales from Corporación Favorita, a large grocery retailer located in Ecuador. The target time series are the total sales for a product family at a particular store on a given day. Future-known covariates are: (1) the total number of items in a product family that were being promoted at a store on a given day, and (2) whether the given day is a holiday. The number of promoted items is log-transformed, and the holiday indicator is binary encoded. In addition, we include day-of-week and day-of-month features as future-known covariates encoded with sine and cosine functions, yielding a total of six future-known covariates. As the Ecuadorian economy is highly dependent on oil, the daily oil price is included as a covariate for the past time steps. The oil price is standardized using mean and standard deviation statistics computed from the training set.

Additionally, the dataset includes the product family, store number, city, state and type as well as a cluster variable, which groups similar stores, as categorical static covariates for each time series.

We use the preprocessed dataset from Shchur et al. [2025] and also use the same horizon of 28 days. We hold out a single evaluation window consisting of the last 28 days of all series as the test set and the preceding 28 days as the validation set. We fix the context length to 56 days. We train all models to minimize an MSE loss in log-transformed space as the primary forecasting loss. As in the related Kaggle competition [Cook et al., 2021], we evaluate performance using the Root Mean Squared Logarithmic Error (RMSLE): $\mathcal{R} = \sqrt{\frac{1}{DH}\sum_{d=1}^{D}\sum_{h=1}^{H}\Big(\log(1+y_{k_d+h}) - \log(1+\hat{y}_{k_d,k_d+h})\Big)^2}$.

Bouc-Wen Hysteretic System. The Bouc-Wen benchmark [Schoukens and Noël, 2017, Noël and Schoukens, 2020] provides two noiseless datasets, originally intended for system identification: (a) a random phase multisine input dataset containing a steady state period of 8192 timesteps and (b) a sine-sweep input dataset in which the sine input signal increases frequency in a frequency band from 20 to 50 Hz. This dataset contains about 153,000 timesteps.

We use this benchmark as a controlled test bed for forecasting with exogenous inputs and define the following two forecasting tasks:

1. **Multisine:** We employ a 70%/10%/20% split on dataset (a) and extract samples using a stride (step size) of 1.
2. **Transfer:** We train on the sinesweep dataset (b) of which we hold out the last 15% as a validation set and evaluate on the multisine dataset (a). Samples are extracted using a stride of 24. This setting is designed to test generalization under a shift in the exogenous forcing process.

For both tasks, we optimize the mean absolute error loss, use a context length of 288 steps, and a forecast horizon of 144 steps. Both datasets are standardized using mean and standard deviation statistics computed from the respective training sets, and performance is reported in standardized units.

Dataset availability. We do not redistribute or re-upload any of the datasets used in this work. The electricity price forecasting (EPF) datasets [Lago et al., 2021] are available from Zenodo[4] the record lists the license as "Other (Open)", with source-specific terms applying to the original electricity-market data. The Favorita store-sales data were accessed through the `fev_datasets` collection [Shchur et al., 2025] on HuggingFace[5], which identifies the original source as Kaggle's *Store Sales – Time Series Forecasting* competition[6] [Cook et al., 2021]; the Hugging Face record is licensed as "other" and points users to the original source terms, while Kaggle permits non-commercial use under the competition rules. The Bouc-Wen hysteresis benchmark data [Schoukens and Noël, 2017, Noël and Schoukens, 2020] are available from 4TU.ResearchData[7] under the Creative Commons Attribution-ShareAlike 4.0 International license (CC BY-SA 4.0).

F.2 Additional information on baselines and training methodology

We consider two groups of baselines: competitive forecasting architectures with exogenous-variable support, and related structured latent-dynamics baselines. For each baseline, we summarize the model class, the implementation used in our experiments, and any adaptations required to match our covariate setting.

TimeXer. The TimeXer model [Wang et al., 2024] is a Transformer-based architecture that is specifically designed for forecasting with exogenous variables. As the original implementation assumes that endogenous and exogenous sequences have the same length, it cannot directly accommodate future-known covariates beyond the historical context. We therefore use the implementation from Li et al. [2026], which modifies the exogenous-input embedding to support future-known covariates in the forecasting setting studied here. However, this version is not able to utilize past-only covariates without further modifications, which only affects the oil price variable in the Favorita task.

[4] `https://zenodo.org/records/4624805`

[5] `https://huggingface.co/datasets/autogluon/fev_datasets`

[6] `https://www.kaggle.com/competitions/store-sales-time-series-forecasting`

[7] `https://data.4tu.nl/articles/_/12967592`

TSMixer. The Time-Series Mixer (TSMixer) [Chen et al., 2023] is an MLP-based architecture built around mixing operations along the time and feature dimensions. We use the implementation from Neuralforecast [Olivares et al., 2022] to run our experiments. TSMixer natively incorporates future-known, past-only and static covariates (in the TSMixerx variant).

GCGNet. The Graph-Consistent Generative Network (GCGNet) [Li et al., 2026] is a recently proposed graph-based time series forecasting model that is specifically designed for forecasting tasks with exogenous variables. We use the original implementation to run baseline experiments. On the Favorita dataset, GCGNet exhibited unstable (internal) variance estimates; we therefore clamp these values to a bounded interval in order to stabilize training on this dataset.

TiDE. Time-series Dense Encoder (TiDE) [Das et al., 2023] is an MLP-based encoder-decoder architecture. We use the implementation from Neuralforecast [Olivares et al., 2022] to run our experiments. TiDE natively incorporates future-known, past-only and static covariates.

SKOLR. Structured Koopman Operator Linear Recurrent Neural Network (SKOLR) [Zhang et al., 2025] is a Koopman-based architecture that is structurally related to our approach. However, it is not able to incorporate future-known exogenous covariates. We use the original implementation and treat the past-only and the past portion of future-known covariates as additional channels of the input and run SKOLR in the non-channel-independent mode.

DLinear. DLinear [Zeng et al., 2023] is a simple linear architecture that includes a seasonal decomposition. We use the implementation from DARTS [Herzen et al., 2022], which natively incorporates past-only, future-known and static covariates.

Koopman-Subnet. Koopman-Subnet [Iacob et al., 2021] is a neural Koopman-with-input architecture originally developed for nonlinear system identification. It learns a lifted latent representation together with a generic MLP-based encoder and a state-dependent input map. We use the original implementation and adapt the inputs to our forecasting setting. Relative to SLiDE, Koopman-Subnet uses a more generic MLP encoder and a state-dependent input map,

CT-Subnet. CT-Subnet [Beintema et al., 2023] is an architecture developed from a continuous-time system-identification perspective. We include it because it is structurally close to our setting in that it learns a latent state from observed history, but it differs from SLiDE in several key respects: it uses a generic MLP encoder, models nonlinear latent dynamics rather than a shared linear latent transport operator, and is motivated by continuous-time identification rather than discrete-time exogenous forecasting. We evaluate CT-Subnet using its original implementation.

MamKO. MamKO [Li et al., 2025] is an adaptive Koopman-style architecture inspired by Mamba [Gu and Dao, 2024] and designed for modeling and control. Unlike SLiDE, which keeps the latent transition fixed and uses the full history to reconstruct a forecast-start latent state, MamKO uses historical context to generate operator-related components while obtaining the initial latent state from only the last target value through an MLP. We include MamKO as a conceptually relevant baseline because it represents an alternative design in which context modulates the latent operator itself rather than improving latent-state estimation at rollout start. We evaluate MamKO using its original implementation. In the original MamKO formulation, the input at time t drives the state transition from t to $t+1$. We additionally evaluate the model with the complete exogenous input sequence shifted forward by one time step. This aligns the forecast covariates with the predicted timestamps and includes the final forecast covariate row in the rollout. Changes that noticeably exceed the reported standard deviation across seeds are improvements in EPEX BE MAE and MSE and EPEX FR MAE, and a further deterioration on the Bouc–Wen transfer task. The overall conclusions and discussions of MamKO remain qualitatively unchanged. We do not evaluate shifted Subnet variants as these architectures support direct feedthrough from $\boldsymbol{u}_t$ to $\boldsymbol{y}_t$.

Baseline Implementations. GCGNet [Li et al., 2026] and TimeXer [Wang et al., 2024] were implemented using the official GCGNet repository[8], which is released under the MIT license. TiDE [Das et al., 2023] and TSMixer [Chen et al., 2023] were run using NeuralForecast[9] [Olivares et al., 2022], which is released under the Apache License 2.0. DLinear [Zeng et al., 2023] was run using Darts[10] [Herzen et al., 2022], which is also released under the Apache License 2.0.

[8] https://github.com/decisionintelligence/GCGNet

[9] https://github.com/nixtla/neuralforecast

[10] https://github.com/unit8co/darts/tree/master

SKOLR [Zhang et al., 2025] was run from its public repository[11], which is released under the MIT license. The CT-Subnet [Beintema et al., 2023] and Koopman-Subnet [Iacob et al., 2021] baselines were implemented using the deepSI toolbox[12], which is released under the BSD 3-Clause license. MamKO [Li et al., 2025] was implemented based on the CC BY 4.0-licensed OpenReview submission and its accompanying supplementary material[13].

Table 9: Usage of past-only, future-known and static exogenous features in baselines. (✓) denotes models that do not natively incorporate static or past features. In this case, we provide an *optional* mechanism for static features that repeats the embedding and concatenates it to the inputs at each time step. For past features, we add their sequence as an additional channel to the target, while keeping a single output channel, if this is possible without major modifications.

Baseline	**Past**	**Future**	**Static**
TimeXer	✗	✓	(✓)
TSMixer	✓	✓	✓
GCGNet	✗	✓	(✓)
TiDE	✓	✓	✓
DLinear	✓	✓	✓
SKOLR	(✓)	✗	(✓)
Koopman-Subnet	(✓)	✓	(✓)
CT-Subnet	(✓)	✓	(✓)
MamKO	(✓)	✓	(✓)

Categorical static embedding. To allow the use of categorical static variables, we adopt the same learnable static embedding scheme for all baselines. For the baselines that do not natively incorporate static variables (TimeXer, GCGNet, SKOLR), we provide an optional mechanism that repeats the embedding and concatenates it to the inputs at each time step. We treat the inclusion of static information as a binary hyperparameter during tuning, allowing these models to leverage the static features if beneficial without forcing a non-native design choice.

Instance stationarization. Differences in instance stationarization can substantially influence forecasting accuracy [Kim et al., 2022]. To isolate the contribution of each method, we fix the instance stationarization protocol across all baselines for a given dataset, choosing the setting that performed best across baseline models. In particular, we apply instance stationarization to both endogenous and exogenous inputs for the EPF-BE and EPF-FR datasets, while applying it only to the endogenous input signal for the remaining EPF datasets and Favorita. We do not apply any instance stationarization to the Bouc-Wen tasks. An exception is Koopman-Subnet, which occasionally diverged at test time, without additional stationarization,in some settings. For this model, we also apply instance stationarization to exogenous inputs on EPF-DE, and to both endogenous and exogenous inputs on the Bouc–Wen transfer task.

Hyperparameter optimization. For all baseline models, we define the hyperparameter search spaces based on the ranges reported in the original papers and list them for the EPF datasets in Table 10. We use a random search with a fixed budget of 30 trials per model and dataset. Each configuration is evaluated on the validation set using early stopping with a patience of five epochs, and the configuration with the best validation performance is selected as the final model setting. The final baseline metrics are reported as averages over five independent runs with different random seeds. For the ablation experiments, we use a smaller random-search budget of 8 trials centered around the best configuration of the base model.

F.3 Computational resources

Experiments were conducted on multiple comparable compute systems. For concreteness, we report one representative configuration used for evaluation: a single node of a high-performance computing

[11] `https://github.com/networkslab/SKOLR`

[12] `https://github.com/MaartenSchoukens/deepSI`

[13] `https://openreview.net/forum?id=hNjCVVmOEQ`

Table 10: Hyperparameter optimization ranges for SLiDE and baselines on all datasets.

Hyperparameter	EPF	Favorita	Bouc-Wen
SLiDE			
batch size	{64, 128, 256}	{256, 512}	{64, 128, 256}
learning rate	$\{10^{-5}, 5 \cdot 10^{-5}, 10^{-4}\}$	$\{5 \cdot 10^{-5}, 10^{-4}\}$	$\{10^{-5}, 5 \cdot 10^{-5}, 10^{-4}\}$
blocks	{2}	{2}	{2}
dynamic dim.	{128, 256}	{128, 256}	{128, 256}
hidden layers	{2}	{2}	{2}
dropout	{0.05, 0.15}	{0.05, 0.15}	{0.05, 0.15}
patch length	{24}	{4, 7}	{4, 8}
TimeXer			
batch size	{16, 32, 64}	{128, 256, 512}	{16, 32, 64}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
feedforward dim. factor	{1, 2, 4, 8}	{1, 2, 4, 8}	{1, 2, 4, 8}
model dim.	{128, 256, 512}	{128, 256, 512}	{128, 256, 512}
encoder layer	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
number of heads	{4, 8}	{4, 8}	{4, 8}
patch length	{24}	{4, 7}	{4, 12, 24}
TSMixer			
batch size	{16, 32, 64}	{64, 128, 256}	{16, 32, 64}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
blocks	{1, 2, 4, 6, 8}	{1, 2, 4, 6, 8}	{1, 2, 4, 6, 8}
dropout	{0, 0.1, 0.3, 0.5, 0.9}	{0, 0.1, 0.3, 0.5, 0.9}	{0, 0.1, 0.3, 0.5, 0.9}
feedforward dim.	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}
GCGNet			
batch size	{16, 32, 64}	{128, 256, 512}	{16, 32, 64}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
feedforward dim.	{64, 128, 256, 512}	{64, 128, 256, 512}	{64, 128, 256, 512}
model dim.	{64, 128, 256, 512}	{64, 128, 256, 512}	{64, 128, 256, 512}
heads	{4}	{4}	{4}
TiDE			
batch size	{512}	{512}	{512}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
hidden size	{256, 512, 1024}	{256, 512, 1024}	{256, 512, 1024}
encoder layer	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
decoder layer	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
decoder output dim.	{4, 8, 16, 32}	{4, 8, 16, 32}	{4, 8, 16, 32}
temporal decoder dim.	{32, 64, 128}	{32, 64, 128}	{32, 64, 128}
dropout	{0, 0.1, 0.2, 0.3, 0.5}	{0, 0.1, 0.2, 0.3, 0.5}	{0, 0.1, 0.2, 0.3, 0.5}
layerNorm	{True, False}	{True, False}	{True, False}
DLinear			
batch size	{32, 64, 128, 256, 512}	{64, 128, 256, 512}	{32, 64, 128, 256, 512}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
kernel size	{24}	{7}	{4, 12, 24, 48}
constant initialization	{True, False}	{True, False}	{True, False}
SKOLR			
batch size	{16, 32, 64}	{128, 256, 512}	{16, 32, 64}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
blocks	{2, 3, 4, 8}	{2, 3, 4, 8}	{2, 3, 4, 8}
hidden layers	{ 1, 2, 3 }	{ 1, 2, 3 }	{ 1, 2, 3 }
dynamic dim.	{ 128, 256, 512 }	{ 128, 256, 512 }	{ 128, 256, 512 }
dropout	{ 0.0, 0.05, 0.1, 0.2 }	{ 0.0, 0.05, 0.1, 0.2 }	{ 0.0, 0.05, 0.1, 0.2 }
patch length	{12, 24}	{4, 7}	{4, 8}
CT-Subnet			
batch size	{64, 128, 256}	{128, 256, 512}	{64, 128, 256}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$
latent dim.	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}
feedthrough	{True, False}	{True, False}	{True, False}
Koop-Subnet			
batch size	{64, 128, 256}	{128, 256, 512}	{64, 128, 256}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$
latent dim.	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}	{16, 32, 64, 128, 256}
feedthrough	{True, False}	{True, False}	{True, False}
u-dependent input map	{True, False}	{True, False}	{True, False}
MamKO			
batch size	{64, 128, 256}	{256, 512}	{64, 128, 256}
learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$	$\{10^{-2}, 10^{-3}, 10^{-4}\}$
kernel size	{2, 24, 48}	{7, 14, 21}	{18, 36, 72}
delta (factor of latent)	{0.25, 0.5, 1}	{0.25, 0.5, 1}	{0.25, 0.5, 1}
latent dim.	{ 16, 32, 64, 128}	{ 16, 32, 64, 128}	{ 16, 32, 64, 128}
hidden dim.	{ 64, 128, 256 }	{ 64, 128, 256 }	{ 64, 128, 256 }
learning rate scheduler	{True, False}	{True, False}	{True, False}

system with four NVIDIA A100 GPUs ($40\,\mathrm{GB}$), an Intel Xeon Platinum 8368 processor, and $512\,\mathrm{GB}$ RAM. Note that each individual training run used a single GPU. The remaining runs were performed on machines with comparable hardware configurations, which is not expected to materially affect

the reported results. The computational efficiency runs were all made on the same hardware for comparability. Overall, the experiments required approximately 1,000 GPU hours.

G Performance of Koopman/system-identification baselines across benchmarks

We observe that Koopman-Subnet, CT-Subnet, and MamKO perform most favorably in settings that closely resemble classical system identification, with compact dynamics and directly observed forcing signals. This is visible in the Bouc–Wen experiments, where the methods perform strongly on Multisine, but degrade substantially under the distribution shift of the Transfer task and when future exogenous inputs are perturbed.

To test whether this trend extends beyond the original benchmarks, we additionally evaluate on the real-world UCI Bike Sharing dataset [Fanaee-T, 2013] and a synthetic Duffing oscillator driven by band-limited Gaussian forcing.

Table 11: Results of Koopman/system-identification baselines on two additional dataset. Best results are highlighted in **bold**.

	Bike		**Duffing**	
Model	**MAE**	**MSE**	**MAE**	**MSE**
SLiDE	**0.230**	**0.129**	0.204	0.089
TimeXer	0.301	0.239	0.218	0.091
GCGNet	0.259	0.174	0.386	0.283
CT-Subnet	0.267	0.162	**0.119**	**0.029**
Koop.-Subnet	0.315	0.211	0.383	0.248
MamKO	0.327	0.262	0.641	0.665

The additional results shown in Table 11 follow this pattern. On Bike Sharing, the system-identification methods show no systematic advantage. CT-Subnet remains competitive, while SLiDE performs best and Koopman-Subnet and MamKO trail the forecasting baselines. On Duffing, which more closely matches a controlled system-identification setting, CT-Subnet performs strongly. At the same time, the weaker Duffing results of Koopman-Subnet and MamKO show that this is not a uniform advantage of all system-identification or control methods.

H Statistical evaluation

Statistical Test Setup. We assess statistical significance using one-sided pairwise tests. For all datasets except Favorita, we apply the one-sided Diebold–Mariano (DM) test on the time series of error differences per sample averaged over seeds. We set the truncation of the variance estimate to the horizon divided by the step size between test samples. We implemented the DM test in Python following the implementation of the R forecast package [Hyndman and Khandakar, 2008]. As each Favorita test sample corresponds to a different product group predicted over the same time range, we instead use a one-sided Wilcoxon signed-rank test on per-product-group error differences, again averaged over seeds. We deem results significant at $p < 0.05$.

Bootstrap Confidence Intervals. In addition, we report 95% confidence intervals of the loss differential (again averaged over the five seeds) obtained via stationary bootstrap applying automatic block-length selection depending on autocorrelation in Table 12. The bootstrap confidence intervals show that SLiDE's advantage is broad but not uniform, with the strongest and most statistically stable gains appearing on EPEX-DE and especially on the Bouc–Wen transfer task.

Variability Across Random Seeds. To assess the stability of the evaluated methods, we report the standard deviation of performance across five random seeds for each baseline–dataset combination in Table Table 13.

Table 12: Bootstrap confidence intervals of the performance differences between SLiDE and baselines. The reported numbers represent the lower and upper bounds of the 95% confidence intervals.

Model	EPF Benchmark										Favorita	Bouc-Wen System			
	NP		PJM		EPEX BE		EPEX FR		EPEX DE		Sales	Multisine		Transfer	
	ΔMAE	ΔMSE	ΔMAE	ΔMSE	ΔMAE	ΔMSE	ΔMAE	ΔMSE	ΔMAE	ΔMSE	ΔMSLE	ΔMAE	ΔMSE	ΔMAE	ΔMSE
SLiDE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TimeXer	0.017 0.004	0.033 0.008	0.001 -0.007	0.001 -0.010	0.003 -0.006	0.007 -0.006	0.005 -0.007	0.053 -0.003	0.021 0.005	0.053 0.012	0.060 0.018	0.082 0.069	0.033 0.025	0.145 0.099	0.158 0.106
TSMixer	0.016 0.003	0.039 0.001	0.005 -0.003	0.002 -0.006	0.009 -0.001	0.006 -0.083	0.057 -0.001	0.028 -0.026	0.034 0.006	0.076 0.006	0.009 -0.003	0.012 0.002	0.003 0.000	0.265 0.224	0.368 0.312
GCGNet	0.015 0.005	0.024 -0.001	0.003 -0.003	0.004 -0.002	0.029 0.014	0.033 -0.048	0.049 0.003	0.019 -0.048	0.031 0.011	0.048 0.007	0.088 0.030	0.141 0.129	0.078 0.071	0.175 0.127	0.210 0.153
TiDE	0.042 0.026	0.046 0.021	0.017 0.002	0.048 0.000	0.034 0.016	0.030 -0.056	0.082 0.004	0.081 -0.025	0.046 0.017	0.071 0.015	0.009 0.000	0.066 0.050	0.022 0.016	0.212 0.164	0.275 0.214
DLinear	0.050 0.028	0.061 0.030	0.017 0.002	0.023 0.001	0.049 0.010	0.059 -0.071	0.076 0.003	0.094 -0.008	0.069 0.021	0.125 0.024	0.088 0.032	0.749 0.667	1.050 0.857	1.089 1.025	2.887 2.674
SKOLR	0.069 0.041	0.094 0.048	0.040 0.019	0.064 0.014	0.034 0.016	0.093 0.007	0.047 0.020	0.098 0.011	0.168 0.123	0.331 0.180	0.086 0.024	0.710 0.629	0.940 0.761	0.507 0.463	0.821 0.761
Koop.-Subnet	0.053 0.030	0.081 0.023	0.034 0.016	0.056 0.008	0.026 0.012	0.018 -0.039	0.066 0.005	0.051 -0.009	0.061 0.035	0.105 0.047	0.022 0.012	-0.013 -0.028	-0.003 -0.007	0.449 0.402	0.721 0.628
CT-Subnet	0.034 0.020	0.029 0.010	0.027 0.012	0.040 0.006	0.042 0.019	0.043 0.001	0.077 0.017	0.119 0.007	0.071 0.038	0.107 0.034	0.012 0.004	-0.008 -0.021	0.000 -0.004	0.496 0.447	0.792 0.700
MamKO	0.045 0.027	0.064 0.032	0.021 0.010	0.031 0.004	0.031 0.019	0.023 -0.014	0.039 0.011	0.080 -0.121	0.083 0.043	0.152 0.052	0.089 0.041	0.402 0.356	0.360 0.296	0.689 0.607	1.787 1.391

Table 13: Standard deviation of performance across five random seeds for each baseline and dataset. We report the population standard deviation, computed over the five runs. Lower values indicate greater stability across random seeds.

Model	EPF Benchmark										Favorita	Bouc-Wen System			
	NP		PJM		EPEX BE		EPEX FR		EPEX DE		Sales	Multisine		Transfer	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	RMSLE	MAE	MSE	MAE	MSE
SLiDE	0.210 ±0.002	0.167 ±0.003	0.157 ±0.003	0.064 ±0.004	0.231 ±0.001	0.367 ±0.003	0.169 ±0.003	0.329 ±0.003	0.265 ±0.002	0.182 ±0.003	0.419 ±0.002	0.085 ±0.001	0.012 ±0.000	0.296 ±0.010	0.139 ±0.010
TimeXer	0.220 ±0.002	0.186 ±0.005	0.154 ±0.001	0.060 ±0.001	0.230 ±0.003	0.367 ±0.007	0.169 ±0.001	0.347 ±0.006	0.278 ±0.002	0.213 ±0.003	0.461 ±0.005	0.160 ±0.001	0.040 ±0.001	0.419 ±0.004	0.273 ±0.004
TSMixer	0.219 ±0.004	0.184 ±0.004	0.158 ±0.002	0.063 ±0.001	0.235 ±0.004	0.335 ±0.014	0.189 ±0.005	0.331 ±0.017	0.283 ±0.005	0.218 ±0.007	0.422 ±0.002	0.092 ±0.004	0.013 ±0.001	0.541 ±0.092	0.480 ±0.169
GCGNet	0.220 ±0.006	0.177 ±0.011	0.157 ±0.003	0.065 ±0.003	0.252 ±0.014	0.367 ±0.021	0.189 ±0.004	0.320 ±0.006	0.285 ±0.006	0.206 ±0.012	0.482 ±0.005	0.220 ±0.086	0.086 ±0.075	0.448 ±0.027	0.322 ±0.040
TiDE	0.243 ±0.006	0.200 ±0.005	0.165 ±0.004	0.081 ±0.009	0.256 ±0.006	0.362 ±0.015	0.200 ±0.001	0.353 ±0.010	0.295 ±0.002	0.220 ±0.001	0.424 ±0.003	0.143 ±0.003	0.031 ±0.001	0.485 ±0.094	0.385 ±0.158
DLinear	0.249 ±0.002	0.211 ±0.003	0.166 ±0.000	0.074 ±0.000	0.258 ±0.006	0.371 ±0.011	0.198 ±0.001	0.363 ±0.004	0.307 ±0.001	0.248 ±0.001	0.483 ±0.001	0.793 ±0.011	0.966 ±0.031	1.354 ±0.204	2.919 ±0.852
SKOLR	0.264 ±0.001	0.237 ±0.001	0.185 ±0.001	0.097 ±0.003	0.256 ±0.002	0.405 ±0.004	0.201 ±0.003	0.372 ±0.007	0.410 ±0.003	0.432 ±0.006	0.478 ±0.007	0.754 ±0.008	0.862 ±0.024	0.782 ±0.005	0.929 ±0.013
K.-Subnet	0.251 ±0.007	0.214 ±0.010	0.181 ±0.003	0.089 ±0.002	0.250 ±0.003	0.360 ±0.004	0.196 ±0.004	0.345 ±0.011	0.312 ±0.004	0.257 ±0.008	0.438 ±0.007	0.064 ±0.005	0.007 ±0.001	0.721 ±0.040	0.815 ±0.088
CT-Subnet	0.237 ±0.002	0.185 ±0.007	0.175 ±0.002	0.082 ±0.004	0.260 ±0.008	0.391 ±0.017	0.208 ±0.003	0.375 ±0.007	0.318 ±0.003	0.247 ±0.004	0.428 ±0.003	0.070 ±0.007	0.010 ±0.002	0.767 ±0.002	0.885 ±0.005
MamKO	0.245 ±0.003	0.213 ±0.005	0.172 ±0.002	0.079 ±0.002	0.256 ±0.003	0.374 ±0.005	0.192 ±0.001	0.324 ±0.012	0.325 ±0.001	0.276 ±0.005	0.489 ±0.011	0.464 ±0.015	0.340 ±0.020	0.944 ±0.135	1.722 ±0.764

I Extended related work

Deep Learning for forecasting with exogenous variables. Deep learning has become an important paradigm for time series forecasting, and explicitly incorporating exogenous variables to capture external drivers of temporal dynamics into neural forecasters remains an active research area. DeepAR represents an early recurrent approach that conditions probabilistic forecasts on past observations together with covariates [Salinas et al., 2020]. Subsequent work has adopted transformer-based architectures for forecasting with exogenous information. The Temporal Fusion Transformer [Lim et al., 2021] introduced attention and gating mechanisms together with explicit handling of static and time-varying covariates. The TimeXer architecture [Wang et al., 2024] further explored transformer-based forecasting and includes covariates with a cross-attention mechanism. A parallel line of work explored

simpler feedforward designs that remain competitive for forecasting with exogenous variables. The Time-series Dense Encoder (TiDE) [Das et al., 2023] adopts a lightweight encoder–decoder formulation. TSMixer [Chen et al., 2023] is another all-MLP architecture that uses mixing operations across time and channels. N-BEATSx [Olivares et al., 2023] extends the N-BEATS architecture [Oreshkin et al., 2020] to forecasting with exogenous covariates. Recently, GCGNet [Li et al., 2026] has been proposed as a generative, graph-based model that explicitly targets the future-known covariate setting. In contrast to attention-, graph-, and MLP-based approaches that primarily rely on flexible feature fusion, SLiDE imposes a constrained input-driven latent dynamics model, which we show improves accuracy and efficiency while promoting generalization under shifted input processes.

Koopman with inputs and controls. The Koopman operator is extensively used to describe dynamical systems. Some works within this community consider additional inputs, such as (Extended) Dynamic Mode Decomposition with control (DMDc/EDMDc) [Proctor et al., 2016, Korda and Mezić, 2018] and Koopman-with-input formulations [Proctor et al., 2018, Kaiser et al., 2021]. These methods were developed primarily for system identification and control, where inputs are typically treated as observed, externally specified control or forcing signals and, in control applications, as directly manipulable. In contrast, forecasting applications involve *exogenous variables* that act as external drivers rather than controls and are often partially observed. Beyond classical approaches with hand-chosen lifting functions, a substantial body of work has studied learning Koopman observables directly from data using neural networks. Takeishi et al. [2017] and Yeung et al. [2019] both learn nonlinear lifting functions and then fit linear dynamics in the lifted space. The first is more explicitly motivated by learning invariant subspaces for DMD and spectral analysis, while the second is framed as deep dictionary learning to avoid manually designing EDMD observables. Related deep operator learning approaches have also been proposed in molecular kinetics, including time-lagged autoencoders and VAMPnets [Mardt et al., 2018, Wehmeyer and Noé, 2018]. Lusch et al. [2018] developed parsimonious, interpretable neural representations of Koopman eigenfunctions by embedding the dynamics on a low-dimensional manifold using an autoencoder architecture. A central difficulty is constructing or learning suitable lifted observables in the presence of inputs, particularly while preserving causal structure [Selby and Asada, 2021]. Recent neural system-identification approaches with inputs, such as Koopman-Subnet [Iacob et al., 2021] and CT-Subnet [Beintema et al., 2023], show that structured latent models can be effective for nonlinear dynamics. Adaptive Koopman-style architectures such as MamKO [Li et al., 2025] use historical context to modulate latent operators, combining Koopman-inspired modeling with Mamba [Gu and Dao, 2024]. However, these methods are primarily motivated by system identification or control rather than multi-step forecasting with exogenous variables. Our work builds on Koopman-with-input formulations by shifting the emphasis from lifting construction to forecasting architecture. Treating the lift as a learnable component lets us focus on two key design questions for exogenous forecasting: how to reconstruct a forecast-start latent state from past context, and how to roll that state forward under future-known exogenous drivers. This yields a structured forecasting architecture that turns Koopman-inspired latent modeling into a practical multi-step prediction design.

Koopman-inspired forecasting and structured linear sequence models. Recent work has adapted the ideas from Koopman theory of learning a latent space in which nonlinear dynamics evolve approximately linearly to forecasting settings. Autoencoder architectures have been proposed to learn latent representations for Koopman-inspired sequential data forecasting [Azencot et al., 2020]. Spectral approaches have likewise been explored for forecasting quasi-periodic dynamical systems [Lange et al., 2021]. The Koopman Neural Forecaster [Wang et al., 2023] is a Koopman-based architecture designed for time series that exhibit temporal distribution shifts. Koopa [Liu et al., 2023] addresses nonstationarity by decomposing time series into time-varying and time-invariant components. In parallel, renewed interest in linear recurrent dynamics provides a complementary perspective. Structured linear RNN models can match or exceed more complex recurrent baselines on long-context problems [Orvieto et al., 2023], while SKOLR [Zhang et al., 2025] provided an explicit connection between a linear recurrent architecture and Koopman theory. More broadly, state-space models such as S4 [Gu et al., 2022] and Mamba [Gu and Dao, 2024] show that structured latent linear dynamics can serve as highly effective sequence backbones. Hybrid approaches such as MamKO [Li et al., 2025] further connect this state-space modeling perspective with Koopman-style latent operators. Unlike such general-purpose or control-oriented sequence models, SLiDE is designed specifically for exogenous multi-step forecasting. In particular, the Koopman forecasting and structured sequence models considered above do not jointly address the setting in which heterogeneous future-known exogenous covariates explicitly drive the latent dynamics throughout the forecast horizon.

NeurIPS Paper Checklist

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 abstract and introduction clearly state the core claims: introducing SLiDE, improving forecasting with exogenous inputs, retaining computational efficiency, and supporting the structured design with ablations. These claims are generally reflected in the reported experiments in Section 6 and the appendices.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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 paper contains an explicit "Conclusion and Limitations" section discussing reliance on approximately linear latent dynamics, possible mismatch for richer latent structures, and sensitivity to imperfect exogenous inputs.

   Guidelines:

   - The answer [N/A] 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: [N/A]

Justification: The paper does not include new theoretical results.

Guidelines:

- The answer [N/A] 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: The manuscript and appendix provide the details needed to reproduce the main results, including the full model specification, datasets, covariate handling, baseline implementations, hyperparameter search protocol and search spaces.

Guidelines:

- The answer [N/A] 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: All datasets used in this paper are publicly available and cited in the manuscript. We provide information on splits, metrics, and training settings in the paper and appendix. The code will be made publicly available upon publication and can be shared with reviewers.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: The main paper specifies key training details such as datasets, evaluation metrics and the hyperparameter optimization protocol. Further experimental details are provided in the appendix, including horizons, context lengths, covariate handling, baseline implementations, hyperparameter search spaces, and additional training details.

   Guidelines:

   - The answer [N/A] 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 reports statistical significance using one-sided Diebold–Mariano tests for most datasets, a Wilcoxon signed-rank test for Favorita, and 95% stationary-bootstrap confidence intervals in Appendix H

   Guidelines:

   - The answer [N/A] 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: Information on the used resources is provided in Appendix F.3.

   Guidelines:

   - The answer [N/A] 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: This research is inline withe the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] 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: [N/A]

    Justification: We do not believe this work represents significant potential societal impacts, either positive or negative.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] 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: Used datasets and baselines are credited throughout the paper. We provide a dataset availability and a baseline implementation paragraph in the appendix, pointing to the respective URLs and mentioning the respective Licenses if possible.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] 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: [N/A]

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

    Guidelines:

    - The answer [N/A] 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: [N/A]

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

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.