COMPKOOP: COMPACT OBSERVABLE COMPLETION FOR PROJECTED KOOPMAN PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Observable closure presents a central challenge for long-horizon prediction with finite-dimensional Koopman models. For many nonlinear autonomous systems, a compact observable space cannot support closed linear evolution under a single finite-dimensional operator. A common approach is therefore to make the finite operator state dependent, allowing unresolved nonlinear structure to be absorbed into adaptive operator coefficients. This work investigates whether such missing structure can instead be represented through a compact learned expansion of the observable space while retaining a time-invariant projected mapping. We propose CompKoop, a Koopman framework that learns a compact product expansion of the observable space for nonlinear state prediction with a time-invariant projected readout. For state prediction, CompKoop targets the Koopman image of the physical-state observable rather than requiring closed linear evolution of the entire lifted representation, leading to a projected-closure formulation for multi-step forecasting. We further show that a class of state-dependent projected mappings can be rewritten as fixed mappings over product-expanded observables, establishing a direct connection between adaptive finite operators and observable completion. Experiments on Lorenz63 and a state-dependent Duffing system demonstrate strong long-horizon prediction with a compact parameterization. Additional experiments demonstrate accurate ten-step prediction of POD-reduced two-dimensional Navier–Stokes dynamics. Analysis of independently trained state-dependent Koopman models shows that product expansion captures their adaptive projected increments. Controlled feature probes and native-readout analysis further show that the learned product observables improve the representation of state evolution and contribute dynamical interactions to the trained predictors. These results support observable completion as a structured alternative to absorbing finite-dimensional closure deficiency through operator adaptation.

1 INTRODUCTION

Koopman theory provides an exact linear description of nonlinear dynamical systems by lifting state evolution to an infinite-dimensional space of observables (Koopman, 1931; Mezic, 2005). In practice, Koopman models use finite-dimensional approximations (Williams et al., 2015; Li et al., 2017). The difficulty is not simply numerical: many nonlinear systems admit no exact global finite-dimensional linear embedding that retains their state dynamics (Brunton et al., 2016a; Kvalheim & Arathoon, 2026; Liu et al., 2025). A compact trial space is therefore generally not exactly closed under the Koopman action, leaving nonlinear structure that must be accommodated by the observable representation, the finite propagator, or both.

One successful strategy is to place additional modeling flexibility in the finite propagator. State- or parameter-dependent Koopman models replace a single fixed matrix with a family of finite operators whose coefficients vary with the current dynamical context. For example, Lusch et al. (2018) used state-dependent spectral parameters to capture frequency variation with a compact representation. More generally, such adaptive finite operators can represent nonlinear evolution that is difficult to capture with a compact fixed operator by allowing unresolved structure to be absorbed into adaptive coefficients. This motivates a complementary question: to what extent can structure represented through operator adaptation instead be exposed explicitly in the observable space, without returning to the large explicit expansions that operator adaptation can avoid? Adaptive coefficients and additional

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

observables need not always represent fundamentally different dynamics; for some finite-dimensional mappings, they provide alternative ways of allocating the nonlinear structure required for prediction. We investigate whether this connection can be exploited to shift modeling capacity from operator adaptation toward a compact learned observable representation while retaining a time-invariant projected readout.

We study this question at the level of the projected physical-state evolution. In particular, we distinguish full invariance of a finite lifted space from the physical-state prediction requirement that the Koopman image of the state be representable from that space through a linear projected readout. We refer to the latter as projected closure. Full finite-dimensional Koopman invariance implies projected closure, but not conversely. At this level, adaptive operators and observable expansion provide alternative parameterizations of the same projected state evolution.

The operator-to-observable connection provides a constructive basis for observable completion. If a state-dependent projected operator varies through auxiliary functions $q_\ell(x)$, a mapping of the form

$$\left(W_0 + \sum_{\ell=1}^{m} q_\ell(x) W_\ell\right) c(x)$$

can equivalently be expressed by a fixed linear readout over the product-expanded observables $[c(x), q_1(x)c(x), \ldots, q_m(x)c(x)]$. The practical challenge is to learn a compact set of useful product observables: explicit enumeration can rapidly enlarge the dictionary, and the relevant interactions are generally unknown in advance. This suggests a design principle—move selected state-dependent structure from operator coefficients into a compact, learned expansion of the observable space. Following this principle, we propose **CompKoop**, which learns a compact observable completion for projected state prediction. It jointly learns a base dictionary and a limited set of product observables, exposing nonlinear state-update structure to a time-invariant linear readout. Factorized products of learned combinations of the base observables provide this expansion without enumerating a full product library.

Our experiments connect the observable-completion perspective to the behavior of trained predictors. Product expansions reconstruct adaptive contributions in independently trained Koopman models; controlled feature probes establish the value of learned products; and native-readout analysis identifies their role in the state update. Long-horizon forecasts on Lorenz63 and nonlinear Duffing dynamics, together with ten-step forecasts of POD-reduced two-dimensional Navier–Stokes flow, demonstrate the predictive utility of this compact, inspectable representation.

Our main contributions are as follows:

- We introduce **CompKoop**, which learns a compact product expansion of the observable space for projected Koopman prediction. Joint learning of the base observables and their multiplicative combinations exposes nonlinear state-update structure to a time-invariant linear readout.
- We develop a **projected observable-completion perspective** connecting operator adaptation with observable expansion: a class of state-dependent projected mappings admits an equivalent fixed-readout representation after product expansion, motivating a structured transfer of modeling capacity from adaptive coefficients to observables.
- Using adaptive-increment reconstruction, controlled product-feature probes, and native-readout interventions, we demonstrate how product expansion captures adaptive projected structure and how learned products improve state-evolution reconstruction and contribute to the trained state update. We validate CompKoop on ODE forecasting and reduced-order fluid prediction.

2 Related Work

Finite-dimensional representations and closure. Extended dynamic mode decomposition approximates Koopman evolution on a prescribed trial dictionary (Williams et al., 2015). Neural approaches learn that dictionary from data, either jointly with a finite propagator or through objectives that encourage invariant coordinates (Takeishi et al., 2017; Li et al., 2017; Lusch et al., 2018). Learning the

dictionary adapts the trial space to the data, while exact finite closure also depends on the underlying dynamics and the domain of representation. Brunton et al. (2016a) identify obstructions to finite invariant spaces that retain the physical state; Liu et al. (2025) establish further restrictions on continuous one-to-one linear immersions for systems with multiple limit sets. The observables retained and the distinction between local and global embeddings therefore matter when interpreting finite representations (Kvalheim & Arathoon, 2026). These results motivate specifying which Koopman images a predictor seeks to represent. For physical-state prediction, the relevant images are the state-update functions; invariance of the entire learned dictionary imposes additional requirements. Section 3 develops this distinction through projected closure. Reprojection and course-correction methods also revisit the observable manifold during rollout (van Goor et al., 2023; Fathi et al., 2023). CompKoop uses this established prediction principle to study how a finite dictionary can be completed for physical-state prediction.

Adaptive finite operators. State-dependent spectral parameters can compactly accommodate amplitude-dependent frequency and continuous-spectrum behavior. Lusch et al. (2018) use an auxiliary network to parameterize continuous frequency, allowing a compact representation of systems with continuous spectra. Time-varying and parametric Koopman models extend finite representations through context-dependent propagators (Hao et al., 2024; Guo et al., 2025). In particular, Guo et al. (2025) learn a dictionary together with a family of operators for parameter-dependent dynamics. In these approaches, conditioning carries dynamical variation that a single compact finite linear representation may not express. This makes adaptive coefficients a representational response to finite-dimensional limitations. Our question concerns their relationship to observable expansion. When coefficient functions are expressed in a scalar basis, their action on dictionary coordinates produces additional product observables. Proposition 2 makes this connection explicit at the projected-map level and quantifies the residual for an approximate coefficient expansion. We examine that relationship using an independently trained parametric model, then learn a compact expansion directly for prediction.

Learned observables and structured nonlinear predictors. Polynomial dictionaries and sparse identification make nonlinear interactions explicit (Brunton et al., 2016b), while learned dictionaries can adapt the features themselves. Projected Koopman approximations also have close connections to neural flow-map models (Buzhardt et al., 2025); structured recurrent architectures provide another connection between Koopman representations and sequence prediction (Zhang et al., 2025). CompKoop uses learned products to enrich a finite dictionary for projected state prediction. The operator-to-observable connection motivates this design, and the experiments examine how the resulting expansion represents nonlinear state-update structure.

3 Finite-Dimensional Closure and Observable Completion

3.1 Finite-Dimensional Closure and Projected State Prediction

Consider an autonomous discrete-time system $x_{t+1} = F(x_t)$ on $\mathcal{X} \subseteq \mathbb{R}^n$. The Koopman operator acts on scalar observables as $(\mathcal{U}g)(x) = g(F(x))$, and componentwise on vector observables. For the physical-state observable $g_x(x) = x$, its Koopman image is precisely the state evolution: $\mathcal{U}g_x = F$.

Let $z : \mathcal{X} \to \mathbb{R}^N$ define a finite trial space $\mathcal{Z} = \operatorname{span}\{z_1, \ldots, z_N\}$ containing the state coordinates $\mathcal{G}_x = \operatorname{span}\{x_1, \ldots, x_n\}$. Full lifted closure requires $\mathcal{U}\mathcal{Z} \subseteq \mathcal{Z}$, equivalently $z(F(x)) = K_z z(x)$ for a fixed matrix K_z. For physical-state prediction, *projected closure* requires

$$\mathcal{U}\mathcal{G}_x \subseteq \mathcal{Z}, \qquad F(x) = K_x z(x), \quad K_x \in \mathbb{R}^{n\times N}. \tag{1}$$

These are exact representability conditions on $\mathcal{X}$; a trained finite predictor approximates the projected condition over the data distribution.

Proposition 1 (Full closure implies projected closure)**.** *If $\mathcal{G}_x \subseteq \mathcal{Z}$ and $\mathcal{U}\mathcal{Z} \subseteq \mathcal{Z}$, then $\mathcal{U}\mathcal{G}_x \subseteq \mathcal{Z}$. The converse does not hold in general.*

The implication follows by inclusion. For $F(x) = x^2$ and $\mathcal{Z} = \operatorname{span}\{x, x^2\}$ on $\mathbb{R}$, $\mathcal{U}x = x^2$ belongs to $\mathcal{Z}$, whereas $\mathcal{U}x^2 = x^4$ does not. This distinction identifies which Koopman images the predictor must approximate.

3.2 Operator Adaptation and Product-Expanded Observables

Consider an adaptive projected mapping $\hat{F}(x) = K_\psi(x)c(x)$ with $c(x) \in \mathbb{R}^d$ and $K_\psi(x) \in \mathbb{R}^{n\times d}$. The following proposition gives a constructive connection between adaptive coefficients and product-expanded observables.

Proposition 2 (Product-expanded projected representation)**.** *Suppose that on a region $\Omega \subseteq \mathcal{X}$,*

$$K_\psi(x) = W_0 + \sum_{\ell=1}^{m} q_\ell(x)W_\ell + R(x), \tag{2}$$

where each q_ℓ is scalar and the $W_\ell \in \mathbb{R}^{n\times d}$ are constant. Define

$$\widetilde{c}(x) = [c(x); q_1(x)c(x); \ldots; q_m(x)c(x)], \qquad \widetilde{W} = [W_0, W_1, \ldots, W_m]. \tag{3}$$

Then $K_\psi(x)c(x) = \widetilde{W}\widetilde{c}(x) + R(x)c(x)$. In particular, $R = 0$ gives an exact fixed-readout representation. If $\sup_{x\in\Omega} \|R(x)\|_{\mathrm{op}} \le \epsilon$ and $\sup_{x\in\Omega} \|c(x)\|_2 \le C$, then

$$\sup_{x\in\Omega} \left\| K_\psi(x)c(x) - \widetilde{W}\widetilde{c}(x) \right\|_2 \le \epsilon C. \tag{4}$$

The proof is given in Appendix A. The proposition exposes a representational choice at the projected-map level: nonlinear dependence may reside in coefficients or in observables multiplying those coefficients. The bound quantifies the approximation of the adaptive projected mapping.

The practical issue is dictionary size. Explicit expansion in Eq. (3) uses $(m+1)d$ coordinates; all quadratic products of d base coordinates already require $d(d+1)/2$ terms. Moreover, the useful coefficient functions and interactions are usually unknown. This motivates learning a limited number of product coordinates together with the base dictionary. We use *observable completion* to describe this target-directed enrichment: the goal is to make the projected Koopman image accessible through a compact fixed readout.

4 CompKoop

CompKoop implements observable completion by jointly learning a base dictionary and a compact set of product observables. Motivated by the operator-to-observable connection in Proposition 2, these products enrich the representation available to a shared linear readout for physical-state prediction. Figure 1 illustrates this construction.

4.1 State-Anchored Observable Dictionary

For a state $x \in \mathbb{R}^n$, an MLP $h_\theta : \mathbb{R}^n \to \mathbb{R}^{d_h}$ produces learned coordinates, and

$$c_\theta(x) = [x; h_\theta(x)] \in \mathbb{R}^d, \qquad d = n + d_h. \tag{5}$$

The explicit state block ensures $\mathcal{G}_x \subseteq \mathrm{span}(c_\theta)$, while the neural block can represent nonpolynomial features. It also makes the contributions of state, neural, and interaction coordinates separately accessible after training.

4.2 Learned Product Observables

To learn a compact product expansion, we augment the base dictionary with r observables, each formed by multiplying two learned linear combinations of $c_\theta(x)$:

$$\chi_\theta(x) = \frac{1}{\sqrt{d}}\, s \odot (Ac_\theta(x)) \odot (Bc_\theta(x)), \quad A, B \in \mathbb{R}^{r\times d}, \quad s = \mathrm{sigmoid}(\eta) \in (0,1)^r. \tag{6}$$

The rows of A and B learn the directions to multiply, while r controls the expansion width. Joint optimization with the base dictionary and projected readout adapts these products to the state-prediction target. This parameterization accommodates interactions among physical and neural observables using $2rd$ factor parameters, without constructing all pairwise products. The optional gate supplies a learnable scale with an optional ℓ_1 penalty; setting $s = \mathbf{1}$ gives the ungated variant.

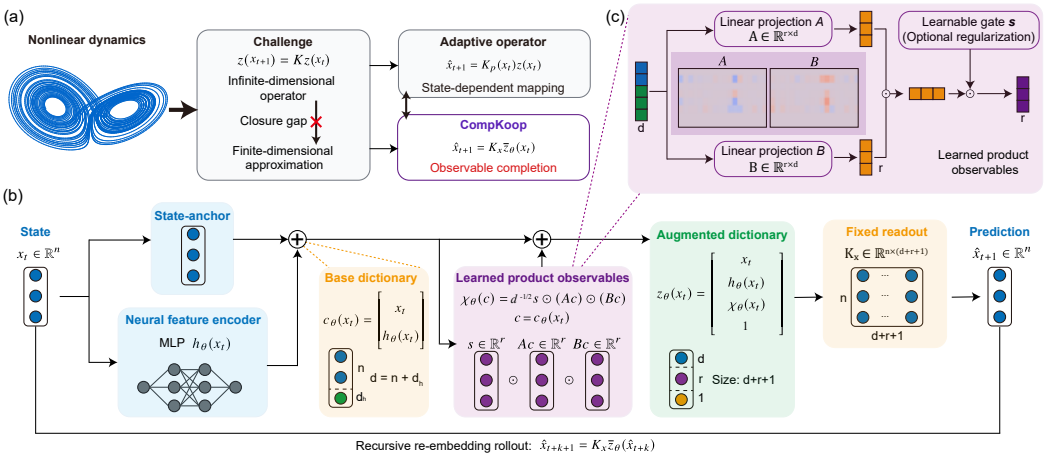


Figure 1: Motivation and architecture of CompKoop. (a) Compact observable spaces generally lack Koopman invariance, motivating operator adaptation or observable expansion to accommodate unresolved nonlinear structure. CompKoop represents selected state-dependent structure through learned product observables and a time-invariant projected readout. (b) CompKoop jointly learns a base dictionary and product observables for physical-state prediction through a shared linear readout. (c) Each product observable multiplies two learned combinations of the base dictionary, providing a compact expansion without enumerating all pairwise products. Circled plus signs denote concatenation.

4.3 Time-Invariant Projected Readout and Recursive Re-Embedding

With $z_\theta(x) = [c_\theta(x); \chi_\theta(x)]$, the residual predictor is

$$\hat{F}_\theta(x) = x + W z_\theta(x) + b, \qquad \hat{x}_{t+k+1} = \hat{F}_\theta(\hat{x}_{t+k}), \quad \hat{x}_t = x_t. \tag{7}$$

Here W and b are shared across all states and time steps. Because $x = P_x z_\theta(x)$ is explicitly retained, this is a linear projected readout on the augmented dictionary $\bar{z} = [z; 1]$: $K_x = [P_x + W, b]$. The direct-prediction variant omits the identity skip. At each rollout step, observables are recomputed from the predicted physical state.

4.4 Training Objective

Training combines one-step error and a closed-loop horizon-H objective,

$$\mathcal{L} = \mathcal{L}_1 + \alpha\mathcal{L}_H + \lambda_s \|s\|_1 + \lambda_{AB}(\|A\|_F^2 + \|B\|_F^2), \qquad \mathcal{L}_H = \frac{1}{H}\sum_{k=1}^{H} \mathrm{MSE}(x_{t+k}, \hat{x}_{t+k}). \tag{8}$$

$\mathcal{L}_1 = \mathrm{MSE}(x_{t+1}, \hat{x}_{t+1})$ and the MSE averages over samples and state coordinates. All coordinates and readout parameters are learned jointly. Multi-step training exposes the model to its own predicted states. Appendix C gives the benchmark-specific dimensions, optimization settings, and checkpoint selection.

5 Experiments

5.1 Experimental Setup and Questions

We evaluate the observable-completion perspective through three linked questions: can product expansion capture adaptive projected increments, do learned products improve the representation of true state evolution, and how does the trained predictor use these coordinates? We then assess recursive prediction and component ablations.

The ODE benchmarks are chaotic Lorenz63 (Lorenz, 1963) and an autonomous Duffing oscillator with state-dependent stiffness, damping, and cubic coefficients. Both use RK4 integration with $\Delta t = 0.01$. The fluid benchmark uses 64 POD coefficients of two-dimensional Navier–Stokes data at viscosity 10^{-5} (Li et al., 2021). Results summarize five training seeds by their mean and sample standard deviation. Mechanism probes use frozen networks and disjoint fitting, validation, and test splits; their errors are normalized by the RMS of the specified target increment. Forecasting uses one-step and rollout NRMSE, valid prediction time (VPT) in simulation time units at normalized-error threshold 0.3, and reconstructed-field relative L^2 error for the fluid task. Appendix B defines these forecasting metrics separately.

5.2 From Adaptive Coefficients to Product Observables

We freeze independently trained state-conditioned ParamKoopmanDL models (PKDL; Guo et al., 2025) on both ODEs. With p the standardized first state coordinate, we isolate the physical-state contribution of $K(p) - K(p_{\mathrm{ref}})$, where p_{ref} is the fitting-state mean. This adaptive increment is reconstructed by fixed ridge readouts over ψ and $[\psi, p\psi, \ldots, p^d\psi]$, for $d = 1, \ldots, 4$. Isolating this contribution focuses the comparison on adaptive variation. Each seed uses 5,000 fitting, 2,000 validation, and 2,000 test states; ridge strength is selected on validation states for each dictionary.

Table 1: Adaptive-increment reconstruction.

Dictionary	Lorenz63	Duffing
ψ	$(3.00 \pm 0.55) \times 10^{-4}$	$(1.90 \pm 0.92) \times 10^{-4}$
$[\psi, p\psi]$	$(2.72 \pm 0.63) \times 10^{-5}$	$(2.42 \pm 0.48) \times 10^{-5}$
$[\psi, p\psi, p^2\psi]$	$(1.22 \pm 0.33) \times 10^{-5}$	$(1.30 \pm 0.53) \times 10^{-5}$
$[\psi, p\psi, p^2\psi, p^3\psi]$	$(5.50 \pm 0.67) \times 10^{-6}$	$(6.23 \pm 2.15) \times 10^{-6}$
$[\psi, p\psi, p^2\psi, p^3\psi, p^4\psi]$	$(3.75 \pm 0.36) \times 10^{-6}$	$(4.61 \pm 1.94) \times 10^{-6}$

Adding $p\psi$ reduces reconstruction error by $90.72 \pm 2.68\%$ on Lorenz63 and $86.06 \pm 3.28\%$ on Duffing; degree-four expansion increases these paired reductions to $98.71 \pm 0.28\%$ and $97.39 \pm 1.10\%$, respectively. Successive product blocks capture adaptive projected structure and further reduce reconstruction error (Table 1). These results instantiate the operator-to-observable connection in Proposition 2 on learned projected mappings in both systems.

5.3 Do Learned Products Improve State-Evolution Features?

We next freeze each CompKoop base dictionary $c = [x; h]$ and fit linear readouts to the true RK4 increment $F_{\Delta t}(x) - x$. We compare c alone, $[c; \chi]$, and c augmented with random factorized products of the same width as χ. Three random factor pairs are evaluated per seed using the same states and ridge-selection procedure as above; their errors are averaged within each seed before aggregation. The product widths are 32 for Lorenz63 and eight for Duffing.

Table 2: State-increment reconstruction.

Dictionary	Lorenz63	Duffing
Base $c = [x; h]$	$(1.15 \pm 0.22) \times 10^{-3}$	$(4.41 \pm 1.44) \times 10^{-2}$
Base + random products	$(8.25 \pm 0.89) \times 10^{-5}$	$(1.23 \pm 0.50) \times 10^{-2}$
Base + learned χ	$(6.06 \pm 0.93) \times 10^{-5}$	$(4.09 \pm 0.79) \times 10^{-3}$

Both random and learned products improve on the base dictionary (Table 2), supporting the use of multiplicative expansion. Learned coordinates further improve on the random-control mean in every seed, with a particularly clear benefit on Duffing. Relative to the base dictionary, the learned expansion reduces error by approximately 95% on Lorenz63 and 90% on Duffing. The results distinguish the value of adding products from the additional benefit of learning their directions within the jointly trained representation. Complementary withheld-observable probes show improved linear accessibility of Lorenz63 products xy and xz (Appendix D).

5.4 How Does the Trained Predictor Use Learned Product Observables?

We decompose the trained residual readout into its state, neural, and product contributions without refitting its weights. Branch outputs are converted to physical coordinates and evaluated on 20,000 test states per seed. For Lorenz63, Table 3 reports five-seed mean coefficients of quadratic projections fitted on 20,000 separate training states (Appendix F).

Table 3: Lorenz63 readout coefficients ($\times 10^3$).

Term	RK4	Neural	Product	Full
$\Delta x : xz$	-0.4627	-0.0706	-0.3920	-0.4627
$\Delta y : xz$	-9.3403	-5.7860	-3.5541	-9.3401
$\Delta z : xy$	$+8.1292$	$+3.9645$	$+4.1635$	$+8.1280$

The product branch contributes to the expected xz and xy interactions, while the complete readout closely matches the corresponding RK4 projections (Table 3). The xz contribution to the first coordinate has a finite-step origin: although $\dot{x} = \sigma(y - x)$ is linear, the second-order term $\frac{1}{2}\Delta t^2 Df(x)f(x)$ contributes $-\frac{1}{2}\sigma\Delta t^2 xz$ to Δx. The trained product branch captures much of this correction. Its quadratic projections achieve test R^2 between 0.983 and 0.997 across outputs and seeds.

Holding all other weights fixed and removing the product output raises Lorenz63 increment errors from approximately 3.5–3.7×10^{-4} to 0.077, 0.401, and 0.195 in the three coordinates. On Duffing, the product branch's velocity output correlates with the total force at 0.934 ± 0.019, and its removal raises velocity-increment error from 0.00504 to 0.4675. Together, physical correspondence and frozen-branch intervention show that the product coordinates participate in the trained state update. The retrained ablations below assess their contribution when the remaining network can adapt.

5.5 Predictive Performance and Ablations

ODE benchmarks. We compare forecasts with PKDL, SKOLR (Zhang et al., 2025), and LSTM. CompKoop uses 8,972 parameters on Lorenz63 and 5,742 on Duffing. Its 256-trajectory test banks use horizons of 1,500 and 10,000 steps, respectively. CompKoop achieves mean VPT 6.567 on Lorenz63 and 48.922 on Duffing, with rollout NRMSE 0.951 and 0.329 (Table 4). Figure 2 illustrates the corresponding trajectory and frequency behavior.

Component ablations. Each variant is retrained and evaluated on the shared ODE test banks. Removing the bilinear branch reduces VPT on both systems, particularly Duffing, where mean VPT falls from 48.922 to 23.651 and rollout NRMSE rises from 0.329 to 0.725 (Table 5). Removing neural coordinates causes larger losses, showing the value of combining learned features with their interactions. The direct-readout variant achieves VPT 6.444 ± 0.297 on Lorenz63 and 41.573 ± 7.509 on Duffing. The ungated variant matches or improves on the full ODE model, showing that the learned product expansion remains effective without gating.

Reduced fluid dynamics. We compare reduced-fluid forecasts with POD-DMD (Schmid, 2010; Tu et al., 2014) and POD-polyEDMD (Williams et al., 2015), which use linear and polynomial observable dictionaries, respectively, over the same POD coefficients. CompKoop obtains ten-step rollout NRMSE 0.2162 ± 0.0037 and reconstructed-field relative L^2 error 0.2494 ± 0.0013, against a POD projection floor of 0.2084. Figure 3 compares the reconstructed flows and spatial errors. This experiment extends prediction to interactions among data-driven modal coordinates. With gate initialization $s_0 = 0.99$ and $\lambda_s = 10^{-3}$, rollout and field errors are 0.2074 ± 0.0029 and 0.2463 ± 0.0010, respectively (Appendix E).

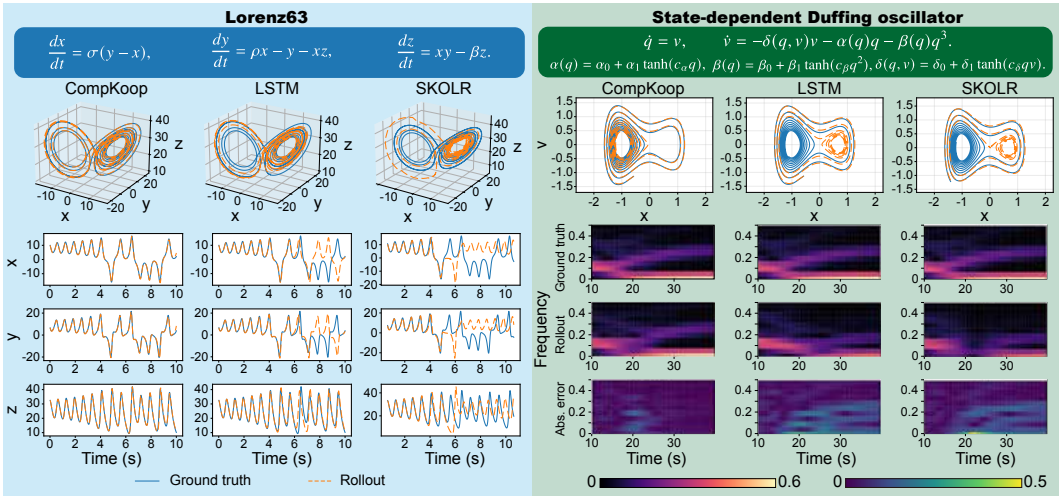


Figure 2: Recursive ODE predictions by CompKoop, LSTM, and SKOLR. Lorenz63 phase portraits and time series show trajectory agreement; Duffing phase portraits and short-time Fourier-transform maps show oscillatory behavior and frequency evolution.

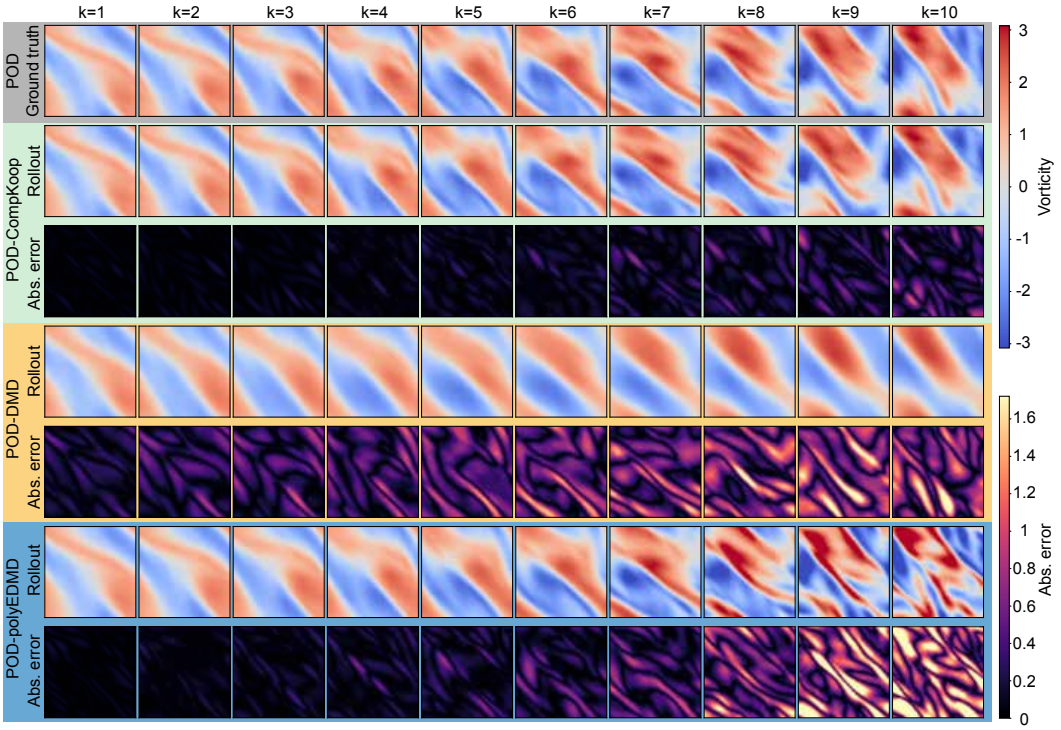


Figure 3: Ten-step Navier–Stokes prediction in POD coordinates. The first row shows the POD-projected reference; each method is shown through its reconstructed vorticity rollout and absolute-error maps. Columns correspond to forecast steps $k = 1, \ldots, 10$. CompKoop is compared with POD-DMD and POD-polyEDMD; multi-seed results and gate ablations are reported in Appendix E.

Table 4: ODE prediction (NRMSE and VPT).

System	Method	1-step ↓	Rollout ↓	VPT ↑
Lorenz63	CompKoop	$(3.06 \pm 0.41) \times 10^{-5}$	0.951 ± 0.020	6.567 ± 0.256
	LSTM	$(2.20 \pm 0.24) \times 10^{-4}$	1.101 ± 0.060	4.206 ± 0.931
	SKOLR	$(9.86 \pm 0.73) \times 10^{-3}$	1.119 ± 0.005	3.924 ± 0.123
	PKDL	$(1.74 \pm 0.21) \times 10^{-4}$	1.031 ± 0.038	3.45 ± 0.20
Duffing	CompKoop	$(1.29 \pm 0.16) \times 10^{-4}$	0.329 ± 0.096	48.922 ± 15.363
	LSTM	$(9.19 \pm 3.97) \times 10^{-4}$	0.985 ± 0.185	17.162 ± 5.037
	SKOLR	$(1.43 \pm 0.22) \times 10^{-2}$	0.633 ± 0.113	30.254 ± 3.871
	PKDL	$(6.97 \pm 0.89) \times 10^{-4}$	0.861 ± 0.171	18.04 ± 2.72

Table 5: ODE component ablations.

	Lorenz63		Duffing	
Variant	VPT ↑	NRMSE ↓	VPT ↑	NRMSE ↓
CompKoop	6.567 ± 0.256	0.951 ± 0.020	48.922 ± 15.363	0.329 ± 0.096
Without bilinear	5.805 ± 0.274	0.990 ± 0.018	23.651 ± 6.036	0.725 ± 0.220
Without neural features	1.939 ± 0.037	1.237 ± 0.004	0.339 ± 0.004	1.062 ± 0.030
Without gate	6.650 ± 0.167	0.936 ± 0.012	56.896 ± 15.285	0.316 ± 0.062

6 Discussion and Limitations

The representation result and mechanism experiments connect operator adaptation to compact observable completion at the projected-state level. Product expansion reconstructs adaptive projected increments, learned products improve linear reconstruction of true state updates, and native-readout interventions identify their contribution to the trained predictor. On Lorenz63 and state-dependent Duffing, these analyses connect the learned expansion to nonlinear interactions in the state update. Retrained ablations and recursive forecasts validate the benefit of combining neural coordinates with learned products, while the reduced-fluid benchmark demonstrates predictive performance in POD coordinates. Together, these findings support observable completion as a structured allocation of nonlinear modeling capacity through a time-invariant projected readout. Future work will connect the representation bound to learning and recursive error accumulation, and extend mechanism analysis to higher-dimensional dynamics.

7 Conclusion

We introduced CompKoop, which jointly learns a compact base dictionary and product observables for projected Koopman prediction with a time-invariant readout. The representation result connects a class of adaptive projected mappings to fixed readouts over product-expanded observables. Adaptive-increment reconstruction verifies this connection in trained models; controlled feature probes and native-readout interventions show that learned products improve state-evolution representation and contribute to the trained update. Long-horizon ODE forecasts and ten-step reduced-fluid predictions demonstrate the effectiveness of the resulting compact representation. These results support observable completion as a structured alternative to carrying nonlinear state-update structure in adaptive operator coefficients.

AI USE STATEMENT

Generative AI tools assisted the preparation of manuscript text and LaTeX source.

REFERENCES

Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLOS ONE*, 11(2):e0150171, 2016a. doi: 10.1371/journal.pone.0150171.

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

Jake Buzhardt, C. Ricardo Constante-Amores, and Michael D. Graham. On the relationship between koopman operator approximations and neural ordinary differential equations for data-driven time-evolution predictions. *Chaos*, 35(4):043131, 2025. doi: 10.1063/5.0249549.

Mahan Fathi, Clement Gehring, Jonathan Pilault, David Kanaa, Pierre-Luc Bacon, and Ross Goroshin. Course correcting Koopman representations. arXiv:2310.15386, 2023. URL `https://arxiv.org/abs/2310.15386`.

Yue Guo, Milan Korda, Ioannis G. Kevrekidis, and Qianxiao Li. Learning parametric koopman decompositions for prediction and control. *SIAM Journal on Applied Dynamical Systems*, 2025. doi: 10.1137/23M1604576.

Wenjian Hao, Bowen Huang, Wei Pan, Di Wu, and Shaoshuai Mou. Deep koopman learning of nonlinear time-varying systems. *Automatica*, 159:111372, 2024. doi: 10.1016/j.automatica.2023.111372.

Bernard O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

Matthew D. Kvalheim and Philip Arathoon. Linearizability of flows by embeddings. *Selecta Mathematica*, 32(2):38, 2026. doi: 10.1007/s00029-026-01136-y.

Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos*, 27:103111, 2017. doi: 10.1063/1.4993854.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

Zexiang Liu, Necmiye Ozay, and Eduardo D. Sontag. Properties of immersions for systems with multiple limit sets with implications to learning Koopman embeddings. *Automatica*, 176:112226, 2025. doi: 10.1016/j.automatica.2025.112226.

Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

Igor Mezic. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41:309–325, 2005.

Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. doi: 10.1017/S0022112010001217.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, 2017.

Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, and J. Nathan Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014. doi: 10.3934/jcd.2014.1.391.

Pieter van Goor, Robert Mahony, Manuel Schaller, and Karl Worthmann. Reprojection methods for Koopman-based modelling and prediction. arXiv:2307.16188, 2023. URL https://arxiv.org/abs/2307.16188.

Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25:1307–1346, 2015.

Yitian Zhang, Liheng Ma, Antonios Valkanas, Boris N. Oreshkin, and Mark Coates. SKOLR: Structured Koopman operator linear RNN for time-series forecasting. arXiv:2506.14113, 2025. URL https://arxiv.org/abs/2506.14113.

A Derivations and Representation Details

A.1 Product-Expanded Representation

For the dictionary and readout in Proposition 2,

$$\widetilde{W}\widetilde{c}(x) = W_0 c(x) + \sum_{\ell=1}^{m} W_\ell q_\ell(x) c(x) \tag{9}$$

$$= \left(W_0 + \sum_{\ell=1}^{m} q_\ell(x) W_\ell \right) c(x). \tag{10}$$

Subtracting this expression from $K_\psi(x)c(x)$ leaves $R(x)c(x)$. The Euclidean induced operator norm then gives

$$\|R(x)c(x)\|_2 \le \|R(x)\|_{\mathrm{op}} \|c(x)\|_2 \le \epsilon C. \tag{11}$$

The coefficient functions need not be polynomials. A polynomial expansion in a scalar conditioning variable, used in Section 5, is one concrete choice. For comparison with the true state evolution, the triangle inequality gives

$$\left\| F(x) - \widetilde{W}\widetilde{c}(x) \right\|_2 \le \|F(x) - K_\psi(x)c(x)\|_2 + \epsilon C. \tag{12}$$

This separates the adaptive model's prediction error from the error of representing its projected mapping with the selected expansion.

A.2 Factorized Interactions and Readout Blocks

Writing the jth rows of A, B as $a_j^\top, b_j^\top$,

$$\chi_j(c) = \frac{s_j}{\sqrt{d}} c^\top a_j b_j^\top c = \frac{s_j}{2\sqrt{d}} c^\top (a_j b_j^\top + b_j a_j^\top) c. \tag{13}$$

Each added coordinate is a factorized quadratic form in the learned base dictionary. The architecture learns which directions to multiply, rather than allocating all quadratic coefficients independently. The residual readout decomposes as

$$\hat{F}(x) - x = W_x x + W_h h_\theta(x) + W_\chi \chi_\theta(x) + b. \tag{14}$$

These blocks expose the branch functions in the fitted representation. Quadratic projections summarize these functions over the sampled state distribution. Feature correlations allow branches to share contributions, while reciprocal rescaling of factors or readout columns leaves the predictor unchanged. A nonzero gate can likewise be absorbed into a factor row; gate regularization controls optimization and preferred scales.

B Benchmark Definitions and Evaluation

B.1 Lorenz63 and State-Dependent Duffing

Lorenz63 uses

$$\dot{x} = 10(y - x), \qquad \dot{y} = x(28 - z) - y, \qquad \dot{z} = xy - \tfrac{8}{3} z. \tag{15}$$

CompKoop is trained on 512 trajectories and validated on 64 trajectories, each of length 2,000, after a 2,000-step burn-in. The internal test split contains 64 trajectories. All ODE simulations use fourth-order Runge–Kutta and $\Delta t = 0.01$.

For Duffing, the state is (q, v) and

$$\dot{q} = v, \qquad \dot{v} = -\delta(q, v)v - \alpha(q)q - \beta(q)q^3, \tag{16}$$

$$\alpha(q) = -1 + 0.6 \tanh(1.2q), \qquad \beta(q) = 1 + 0.8 \tanh(0.8q^2), \tag{17}$$

$$\delta(q, v) = 0.05 - 0.5 \tanh(qv). \tag{18}$$

The dynamics are autonomous although the coefficients depend on state. Initial conditions are sampled from $[-2, 2]^2$; trajectories are filtered using maximum absolute state 8, minimum standard deviation 0.05 in both coordinates, minimum energy 0.01, and a tail-energy ratio threshold 0.15 over the final 20% of the trajectory. Training/validation contain 512/64 trajectories of length 2,000; the internal 64-trajectory test split has length 10,000. Model inputs are standardized with the corresponding training statistics.

B.2 Shared ODE Evaluation and Metrics

The common-bank evaluation uses a fixed generation seed, 20260506, and 256 trajectories per system. The stored lengths are 1,564 for Lorenz63 and 10,064 for Duffing. With zero-based indexing, the initial state is frame 63 and the first prediction targets frame 64. SKOLR receives the preceding 64-frame context; state-input models start from frame 63. All future states are predicted recursively. The rollout lengths are 1,500 and 10,000, respectively.

Let μ_{bank} be the mean of all states in the fixed test bank, H the forecast horizon, and M the number of trajectories. The common evaluator computes

$$D^2 = \frac{1}{MHn} \sum_{i=1}^{M} \sum_{k=1}^{H} \|x_{i,t+k} - \mu_{\mathrm{bank}}\|_2^2, \qquad \mathrm{NRMSE}_{\mathrm{roll}} = \frac{\sqrt{\frac{1}{MHn} \sum_{i,k} \|x_{i,t+k} - \hat{x}_{i,t+k}\|_2^2}}{D}. \tag{19}$$

One-step NRMSE uses only the first predicted state in its numerator and the same D. For VPT,

$$e_i(k) = \frac{\|x_{i,t+k} - \hat{x}_{i,t+k}\|_2}{\sqrt{T^{-1} \sum_{j=1}^{T} \|x_{i,j} - \mu_{\mathrm{bank}}\|_2^2}}, \quad \mathrm{VPT}_i = \min\{k\Delta t : e_i(k) > 0.3\}, \tag{20}$$

where the denominator uses the complete stored trajectory. If no crossing occurs, VPT is $H\Delta t$. First average VPT over trajectories within each trained seed, then report the mean and sample standard deviation over seeds $11, 22, 33, 44, 55$. The cap is 15 time units for Lorenz63 and 100 for Duffing.

C Training and Model Configurations

The CompKoop ODE encoder has two width-64 hidden layers with GELU activations. Table 6 lists the model and training configurations. AdamW uses zero weight decay, gradient clipping at norm 1, and a plateau scheduler with factor 0.5, patience five, and minimum learning rate 10^{-6}. ODE prediction and probe analyses use VPT-selected checkpoints. Fluid checkpoints are selected by validation loss. The default fluid configuration uses gate initialization $s_0 = \sigma(-1)$; Table 11 lists the other gate settings.

Table 6: CompKoop configurations.

Setting	Lorenz63	Duffing	NS POD
State dimension	3	2	64
Encoder hidden widths	64, 64	64, 64	256, 256
d_h / r	32 / 32	16 / 8	128 / 128
Learning rate	10^{-3}	10^{-3}	10^{-3}
Batch size	4,096	4,096	512
Training horizon / α	10 / 0.5	10 / 0.5	10 / 0.2
λ_s / λ_{AB}	0 / 0	0 / 10^{-6}	10^{-3} / 0

The ODE maximum training budget is 200 epochs. Early-stopping patience is 500 for Lorenz63 and 30 for Duffing. CompKoop has 8,972 and 5,742 parameters, respectively, including biases and gates. Computing the bilinear branch requires $O(rd)$ arithmetic in addition to the encoder and readout; it does not materialize a $d \times d$ product matrix.

C.1 Baseline Architectures and Training

Table 7 specifies the ODE baseline architectures. LSTM uses a residual state readout and carries its recurrent hidden state through each forecast. SKOLR operates on a 64-frame context with segment

length eight and predicts 16 frames per call. Its two blocks share width-128 encoders and decoders, with dynamic dimension 128, dropout 0.05, and a learned global spectral mask. Channels are modeled jointly. During recursive evaluation, predicted frames are appended to the context.

PKDL uses the dictionary $\psi(\tilde{x}) = [1; \tilde{x}; \psi_{\mathrm{NN}}(\tilde{x})]$ with 128 learned observables. Its dictionary network and parameter-to-operator network each use two width-128 residual blocks with tanh activations. The scalar condition is the standardized first state coordinate. Training minimizes one-step lifted prediction error, and the state coordinates are extracted from the predicted dictionary.

Table 7: ODE baseline architectures.

Method	Setting	Lorenz63	Duffing
LSTM	Hidden width	256	64
LSTM	Recurrent layers	3	1
SKOLR	Context / prediction / segment	64 / 16 / 8	64 / 16 / 8
SKOLR	Dynamic dimension	128	128
SKOLR	Blocks	2	2
PKDL	Learned observables	128	128
PKDL	Dictionary / operator widths	$128, 128$ / $128, 128$	$128, 128$ / $128, 128$

The ODE baselines use batch size 4,096 and a maximum budget of 200 epochs. LSTM and SKOLR use AdamW at learning rate 10^{-3} with zero weight decay. LSTM uses a ten-step rollout loss with weight 0.5; SKOLR uses four recursive prediction chunks with weight 0.5. PKDL uses Adam at learning rate 2×10^{-4}. Gradient clipping and the plateau scheduler follow the settings above. Early-stopping patience is 500 epochs for Lorenz63 and 30 for Duffing in LSTM and PKDL; SKOLR uses 500 for both systems.

C.2 Hyperparameter Search and Final Settings

Table 8 lists the search ranges, with 36 CompKoop, 54 LSTM, 30 SKOLR, and eight PKDL candidates per system. Candidates are ranked by VPT, with lower rollout NRMSE breaking ties. The selected architecture is held fixed across seeds $11, 22, 33, 44, 55$, using the final training settings specified above.

Table 8: ODE hyperparameter search ranges.

Method	Hyperparameter	Values
CompKoop	d_h; r	8, 16, 32; 8, 16, 32
	α	$0.2, 0.5$
LSTM	Hidden width; recurrent layers	64, 128, 256; 1, 2, 3
	Learning rate; α	$2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}$; $0.2, 0.5$
SKOLR	(s, p, g)	Five tuples specified below
	Dynamic dimension; learning rate	$32, 64, 128$; $5 \times 10^{-4}, 10^{-3}$
PKDL	Observables / network width	Four pairs specified below
	Learning rate	$2 \times 10^{-4}, 10^{-3}$

The CompKoop search uses a two-layer width-64 GELU encoder and varies the neural and product coordinate counts independently. LSTM varies hidden width and recurrent depth. SKOLR searches the sequence tuples $(s, p, g) \in \{(32, 8, 4), (64, 8, 4), (64, 16, 8), (128, 16, 8), (128, 32, 16)\}$, where s, p, g are context, prediction, and segment lengths. PKDL pairs the learned-observable count and network width as $(32, 64)$, $(64, 64)$, $(64, 128)$, or $(128, 128)$; dictionary and operator networks share the same width.

C.3 Parameter Counts and Computational Cost

Table 9 gives the trainable parameter counts for the ODE architectures, counting shared weights once. CompKoop uses fewer than 9,000 parameters on both systems. Its compactness follows from the small neural dictionary and the factorized product branch.

Let $u_0 = n, u_1, \ldots, u_L = d_h$ be the encoder widths, $d = n + d_h$ the base-dictionary dimension, and $N = d + r$ the complete dictionary dimension. With biases in each encoder layer and the readout, the gated architecture has

$$P = \sum_{\ell=1}^{L} (u_{\ell-1} + 1)u_\ell + 2rd + r + n(N + 1). \tag{21}$$

The terms count the encoder, the two product factors, the gate logits, and the projected readout. The identity skip adds no trainable parameters.

Table 9: ODE model parameter counts.

Method	Lorenz63	Duffing
CompKoop	8,972	5,742
LSTM	1,320,707	17,538
SKOLR	105,434	103,378
PKDL	2,313,868	2,279,942

For one state, the dominant arithmetic cost is

$$C_{\text{step}} = O\left(\sum_{\ell=1}^{L} u_{\ell-1}u_\ell + 2rd + nN\right). \tag{22}$$

Elementwise multiplication, scaling, and gating contribute $O(r)$ additional operations. A full homogeneous quadratic library over c contains $d(d+1)/2$ coordinates. CompKoop instead evaluates r factorized interactions, with $2rd$ factor parameters and nr associated readout weights. Thus r directly controls the size and cost of observable completion.

Recursive prediction over horizon H costs $O(HC_{\text{step}})$, including evaluation of the encoder at every predicted state. For batch size B, the rollout training objective requires $O(BHC_{\text{step}})$ forward arithmetic and storage of activations along the unrolled computation for backpropagation. These expressions connect the compact parameterization to its per-step and rollout costs.

D Frozen-Representation Probe Details

Adaptive-increment target. For both systems, let ψ and K denote the frozen PKDL dictionary and operator in its implementation's row-vector convention. We reconstruct

$$\Delta f_p(x) = D\left(\psi(\tilde{x})^\top [K(p) - K(p_{\text{ref}})]\right)_{\text{state}}^\top, \quad \tilde{x} = D^{-1}(x - \mu), \tag{23}$$

where μ, D are the checkpoint's training mean and diagonal standard deviation. The condition p is its standardized first coordinate, and p_{ref} is averaged over fitting states. The target isolates the adaptive physical-state contribution. Polynomial blocks are evaluated at degrees one through four, separately selecting ridge strength for each degree.

The adaptive-increment and true-flow probes use 5,000/2,000/2,000 states from separate train/validation/test trajectory splits. For target y, their normalized error is $\sqrt{\sum_i \|\hat{y}_i - y_i\|_2^2 / \sum_i \|y_i\|_2^2}$. Feature columns are centered and standardized using fitting states, constant columns are removed, and an intercept is fitted. Ridge strength is selected from $\{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$ by validation error. Percentage reductions are computed within each seed before averaging.

True-flow and random-product probes. Targets are physical RK4 increments evaluated at the sampled states. The base encoder is frozen together with the learned product factors. For each of three random controls, entries of $A, B \in \mathbb{R}^{r \times d}$ are independently sampled from $\mathcal{N}(0, 1/d)$, giving $d^{-1/2}(Ac) \odot (Bc)$. Learned and random branches use identical widths; each augmented dictionary receives a separately fitted ridge readout. Random-control errors are averaged within a training seed before computing the five-seed mean and sample standard deviation. This comparison holds the jointly trained base representation fixed.

Withheld-observable target. Lorenz63 probes predict the physical products xy and xz from frozen features. These products do not enter prediction training as feature inputs or auxiliary losses. Fit/validation/test states are sampled from disjoint splits, with counts 50,000/20,000/20,000. The full model and no-neural-feature control are probed with and without their bilinear coordinates. The raw-state xz probe has $R^2 = 0.9678$. Table 10 reports test NRMSE for the raw-state dictionary and the learned dictionaries with and without product augmentation. Product augmentation reduces error by $92.28 \pm 3.11\%$ for xy and $92.88 \pm 2.58\%$ for xz in the full model. These probes use the fitting-target mean as their normalization reference. In the no-neural-feature control, 32 bilinear coordinates can span the six-dimensional space of homogeneous quadratic monomials.

Table 10: Lorenz63 withheld-observable NRMSE.

Model	Dictionary	xy	xz
CompKoop	$[x; h]$	$(7.02 \pm 1.35) \times 10^{-4}$	$(3.59 \pm 1.41) \times 10^{-4}$
CompKoop	$[x; h; \chi]$	$(5.13 \pm 1.46) \times 10^{-5}$	$(2.29 \pm 0.29) \times 10^{-5}$
No MLP	x	0.960	0.179
No MLP	$[x; \chi]$	$(5.53 \pm 0.20) \times 10^{-8}$	$(4.730 \pm 0.005) \times 10^{-8}$

The lower probe errors show that product augmentation makes the dynamical interactions xy and xz more accessible to a linear readout. In the full model, this improvement emerges from training for state prediction, connecting the learned expansion to observables that enter the Lorenz63 dynamics. The no-neural-feature control illustrates the direct algebraic route: products of state coordinates provide a quadratic dictionary from which these interactions can be recovered.

E Reduced Navier–Stokes Results

The vorticity data use $\nu = 10^{-5}$, 1,200 trajectories, 20 frames, and a 64×64 spatial grid (Li et al., 2021). The trajectory split is 1,000 training, 100 validation, and 100 test. The mean field and 64-mode POD basis are fitted on training trajectories, and the predictor operates on standardized POD coefficients. Evaluation starts at frame nine and predicts frames 10–19 recursively. Predicted coefficients are mapped back through the fixed POD basis to compute full-field relative L^2 error over the ten-frame window. The POD projection relative L^2 error is 0.20842.

Table 11 reports one-step and ten-step rollout NRMSE and full-field relative L^2 error for different gate initializations and penalties, holding the other model and training settings fixed. The high-initialized, regularized setting improves all three reported metrics on all five paired seeds relative to the no-gate setting.

Table 11: Navier–Stokes prediction.

Setting	1-step	Rollout	Field L^2
$s_0 = \sigma(-1)$, $\lambda_s = 10^{-3}$	0.05147 ± 0.00068	0.21622 ± 0.00371	0.24939 ± 0.00129
No gate	0.05073 ± 0.00087	0.21286 ± 0.00473	0.24823 ± 0.00163
$s_0 = 0.99$, $\lambda_s = 0$	0.05031 ± 0.00122	0.21002 ± 0.00124	0.24725 ± 0.00042
$s_0 = 0.99$, $\lambda_s = 10^{-3}$	0.04978 ± 0.00077	0.20735 ± 0.00294	0.24634 ± 0.00099

The reduced linear and polynomial references obtain full-field errors 0.441 (POD-DMD) and 0.498 (POD-PolyEDMD) at POD rank 64. Each reference is fitted once; neural-model results summarize five training seeds.

F Native-Readout and Finite-Step Diagnostics

Physical-coordinate branch decomposition. For the residual predictor, the physical state increment is

$$\hat{F}(x) - x = D[W_x \tilde{x} + W_h h(\tilde{x}) + W_\chi \chi(\tilde{x}) + b], \qquad \tilde{x} = D^{-1}(x - \mu). \tag{24}$$

All branch weights remain at their trained values. The sum agrees with direct model evaluation to a maximum reconstruction RMSE of 3.1×10^{-7} across the analyzed checkpoints. Frozen removal subtracts $DW_\chi \chi$ from this output, with no reoptimization. Table 12 reports errors relative to the RMS of each true physical-state increment. Frozen removal quantifies the product branch's contribution to the trained state update; retrained ablations assess its benefit when the remaining network is reoptimized (Table 5).

Table 12: Frozen product-branch removal.

System	Increment	Full	Product removed
Lorenz63	Δx	$(3.75 \pm 0.72) \times 10^{-4}$	$(7.67 \pm 3.62) \times 10^{-2}$
	Δy	$(3.51 \pm 0.68) \times 10^{-4}$	$(4.01 \pm 0.80) \times 10^{-1}$
	Δz	$(3.54 \pm 0.54) \times 10^{-4}$	$(1.95 \pm 0.34) \times 10^{-1}$
Duffing	Δq	$(1.37 \pm 0.47) \times 10^{-3}$	$(6.23 \pm 5.69) \times 10^{-3}$
	Δv	$(5.04 \pm 0.86) \times 10^{-3}$	$(4.67 \pm 2.67) \times 10^{-1}$

Lorenz63 branch projections. We project each native branch and the true RK4 increment onto $[1, x, y, z, x^2, xy, xz, y^2, yz, z^2]$, using 20,000 training states and 20,000 separate test states per seed. The design matrix has condition number approximately 8.7×10^3. Test R^2 for the full model's product-branch projections ranges from 0.983 to 0.997. The summed branch projections reproduce the near-zero true $\Delta x : xy$ coefficient through opposing neural and product contributions of approximately $+5.2 \times 10^{-5}$ and -5.2×10^{-5}.

For a vector field f, the finite-step expansion is

$$F_{\Delta t}(x) - x = \Delta t f(x) + \tfrac{1}{2}\Delta t^2 Df(x)f(x) + O(\Delta t^3). \tag{25}$$

Since $\dot{x} = \sigma(y - x)$, the second-order contribution to Δx contains $-\sigma \Delta t^2 xz/2 = -0.0005xz$. The empirical RK4 projection coefficient is approximately -0.000463, incorporating higher-order effects over the sampled distribution. A separate frozen-feature probe reconstructs the correction $F_{\mathrm{RK4}}(x) - x - \Delta t f(x)$ in its first coordinate: adding learned products reduces target-RMS-normalized error from 2.01×10^{-3} to 1.09×10^{-4}, averaged over five seeds.

Analytic no-neural-feature control. When $c = \tilde{x}$, each product coordinate is quadratic in the physical state. Expanding A, B, s, W_χ and the training standardization gives exact quadratic coefficients for the fitted predictor, without a regression probe (Table 13). Their signs and scales agree with the corresponding leading dynamical terms; the Taylor reference omits higher-order time contributions. The full model's stronger forecasting performance supports combining these state interactions with learned neural coordinates.

Table 13: Lorenz63 coefficients without neural features.

Term	Coefficient	Leading Taylor
$\Delta x : xz$	-0.0005361 ± 0.0000016	-0.0005000
$\Delta y : xz$	-0.0094466 ± 0.0000016	-0.0100000
$\Delta z : xy$	0.0076848 ± 0.0000024	$+0.0100000$

Duffing force alignment. Across five seeds, the product branch's physical velocity output correlates with total force at 0.934 ± 0.019 and with the state-dependent cubic restoring force at 0.848 ± 0.044. Together, force alignment and frozen removal show that the learned product branch contributes to the nonlinear velocity update.

G Frozen-Dictionary Lifted-Update Diagnostic

This diagnostic examines the recursive prediction behavior of the learned dictionary under two update rules: the trained projected readout with re-embedding and an affine update fitted over all lifted coordinates.

We hold the dictionary trained for projected prediction fixed and fit an affine ridge update from $z(x)$ to $z(F(x))$ using 5,000/2,000/2,000 fitting/validation/test states. Ridge strength is selected by one-step validation error over the full dictionary. Starting at 128 sampled test states, the fitted update is iterated 100 times; its first state coordinates are compared with the trained predictor's recursive re-embedding. Table 14 reports final-step state RMSE in training-standardized coordinates.

Table 14: State RMSE after 100 steps.

System	Lifted update	Re-embedding
Lorenz63	0.81407 ± 0.02994	0.00389 ± 0.00232
Duffing	0.24810 ± 0.10770	0.00240 ± 0.00054

For seed 11, increasing the fitting set to 50,000 states and validation/test sets to 10,000 gives final-step errors 0.792 versus 0.00424 on Lorenz63 and 0.215 versus 0.00237 on Duffing.

For this learned dictionary, the trained projected predictor achieves substantially smaller state errors after 100 steps than the fitted lifted update on both systems. The difference persists in the larger-sample fit. These results give an empirical counterpart to the distinction in Section 3: a dictionary's utility for projected state prediction and its behavior under a recursively applied lifted update are separate properties.

H Further Context on Finite Koopman Models

Learning an observable space. EDMD projects the Koopman action onto a chosen trial dictionary (Williams et al., 2015). Dictionary learning adapts that space to data: Li et al. (2017) jointly learn dictionary functions and a finite operator, while Takeishi et al. (2017) learn features through linear-regression residuals in observable space. CompKoop organizes dictionary learning around the images of physical-state observables and uses product coordinates to enrich their linear representation.

Reprojection and recursive prediction. Reprojection methods examine how finite Koopman approximations interact with the manifold of embedded physical states (van Goor et al., 2023). Periodic reencoding revisits the learned representation during inference to improve long-horizon prediction (Fathi et al., 2023). These works relate prediction quality to both representation and rollout. CompKoop recomputes its dictionary from each predicted state; its probes measure the accessibility of state-update terms, and Appendix G examines a fitted update over the entire dictionary.

Structured expansion. Polynomial libraries and sparse regression expose nonlinear interactions through explicit candidate functions (Brunton et al., 2016b). CompKoop learns the directions to multiply among physical and neural coordinates, with expansion width r. Feature probes and native-readout analysis connect this design to the representation and use of dynamical interactions.