EQUIVARIANT LAGRANGIAN NEURAL NETWORKS WITH NOETHER GUARANTEES

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning physical dynamics from data supports simulation and control. Lagrangian Neural Networks (LNNs) learn a scalar Lagrangian in generalized coordinates and derive dynamics through the Euler–Lagrange equations, but standard LNNs do not guarantee spatial symmetries or their conservation laws. We introduce *Equivariant Lagrangian Neural Networks* (EQ-LNNs), which lift nonlinear generalized coordinates and velocities to features with known linear transformation laws and use an equivariant network to produce an invariant scalar Lagrangian. We show that enforcing invariance in the lifted representation makes the resulting Lagrangian invariant to the corresponding transformations of the original generalized coordinates. By Noether's theorem, this guarantees the corresponding conserved quantities when the learned Lagrangian is regular. The construction handles translations, articulated and rigid-body systems, and fixed external fields through their remaining symmetries. Across five systems, EQ-LNN improves over a lifted LNN on four systems, including $2.4\times$ to $52.5\times$ lower median trajectory MAE on the free-body systems. Momentum components tied to enforced symmetries drift by at most 5.7×10^{-4}, while matched lifted LNNs show order-one drift in the corresponding angular-momentum components. EQ-LNN can improve prediction while making the appropriate conservation laws part of the model rather than something inferred from data.

1 INTRODUCTION

Accurate models of physical dynamics are needed across science and engineering, from simulation and prediction to planning and control. In many systems the governing principles are known, but an accurate analytical model still requires detailed knowledge of geometry, inertial parameters, interactions, and external forces that may be unknown or difficult to characterize. Generic neural dynamics models can learn dynamics directly from data, but do not necessarily respect the physical structure that governs the underlying system and may generalize poorly beyond the observed trajectories. Lagrangian neural networks (LNNs) address this problem by learning a scalar Lagrangian $L(q, \dot{q})$ and deriving the dynamics through the Euler-Lagrange equations (Cranmer et al., 2020; Lutter et al., 2019), yielding a physically structured model in generalized coordinates without requiring the equations of motion analytically.

However, a generic LNN does not automatically inherit known spatial symmetries of the underlying system. A free system, for example, should behave the same after a global rotation or translation. Noether's theorem connects each continuous symmetry of the Lagrangian to a conserved quantity: translation symmetry implies conservation of linear momentum, while rotational symmetry implies conservation of angular momentum (Marsden and Ratiu, 1999). If a learned Lagrangian assigns different values to symmetry-related states, these conservation laws are no longer guaranteed. Previous work has addressed this using invariant features, relative particle coordinates, and equivariant neural networks (Müller, 2023; Bhattoo et al., 2022b; Finzi et al., 2021).

One way to impose known symmetries on an LNN is to replace its scalar network with an equivariant architecture. However, standard equivariant networks assume a known linear action on their features, whereas mechanical models often use generalized coordinates such as joint angles or attitude parameters, on which spatial rotations and translations act nonlinearly. We therefore ask: *How can we*

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.

construct a neural Lagrangian that is exactly invariant to known physical symmetries while retaining nonlinear generalized coordinates?

Many systems, however, are not invariant to the full spatial symmetry group. External fields such as gravity break some symmetries while preserving others. A symmetry-aware model should represent these external effects without discarding the remaining symmetry or enforcing a symmetry the physics does not possess. This leads to our second question: *How can the same framework incorporate external fields and complex articulated or rigid-body dynamics while preserving exactly the symmetries that remain?*

We introduce *Equivariant Lagrangian Neural Networks* (EQ-LNNs) to address these challenges. Rather than requiring the generalized coordinates themselves to transform linearly, EQ-LNN maps them to symmetry-compatible configuration and velocity features with known linear group actions, and learns an invariant scalar Lagrangian from those features. We show that if the lifted model is invariant, then the resulting Lagrangian is also invariant under the corresponding transformations of the original generalized coordinates. By Noether's theorem, this gives the associated conservation laws for the regular learned dynamics. The same construction accommodates fixed external fields such as gravity. The same framework covers particle, articulated, and rigid-body systems in compact generalized coordinates.

We compare EQ-LNN with a lifted LNN that receives the same lifted inputs but replaces the equivariant scalar network with an ordinary dense MLP. This controls for the information supplied by the lift and compares the complete EQ-LNN architecture with a conventional lifted LNN under the same data and optimization protocol. EQ-LNN achieves lower median trajectory error on four of five primary systems. We also directly evaluate the drift in the learned momentum quantities: momenta associated with enforced symmetries remain constant to numerical tolerance along the learned flow, while quantities corresponding to broken symmetries are allowed to change. Our contributions are:

- We introduce EQ-LNN, which enforces known physical symmetries while retaining nonlinear generalized coordinates, and prove that invariance in the lifted space gives the corresponding conservation laws for the regular learned dynamics.
- We extend the construction to translations, articulated and rigid-body systems, and fixed external fields that preserve only a subgroup of the original symmetry, and evaluate it against matched lifted baselines, learned conservation laws, and symmetry-related out-of-distribution tests.

2 BACKGROUND AND PROBLEM SETUP

Lagrangian mechanics. Let $\mathcal{Q}$ be a smooth configuration manifold with tangent bundle $T\mathcal{Q}$. A *state* is a tangent vector $(q, \dot{q}) \in T\mathcal{Q}$, where $q \in \mathcal{Q}$ is the *configuration* (or generalized coordinates) and $\dot{q} \in T_q\mathcal{Q}$ is its *generalized velocity*. In local coordinates, a time-independent Lagrangian $L : T\mathcal{Q} \to \mathbb{R}$ defines the Euler–Lagrange equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = 0. \tag{1}$$

We call L *regular* when its velocity Hessian $\partial^2_{\dot{q}\dot{q}} L(q, \dot{q})$ is nonsingular on the states of interest. Under this condition, equation (1) determines a second-order vector field and, hence, a local flow on $T\mathcal{Q}$.

Lagrangian Neural Networks. A Lagrangian Neural Network (LNN) replaces a hand-derived Lagrangian with a parameterized scalar function $L_\theta(q, \dot{q})$ (Cranmer et al., 2020). Accelerations are obtained by solving equation (1) for $\ddot{q}$, and the parameters are trained from trajectory data, for example, by matching accelerations or short rollouts. This variational parameterization is already a useful inductive bias: it is more structured than fitting an arbitrary first-order vector field while remaining more general than models that prescribe a quadratic kinetic energy and a position-only potential (Greydanus et al., 2019; Cranmer et al., 2020). Our non-equivariant reference model is this full reduced-coordinate LNN, with the scalar Lagrangian otherwise unconstrained.

Continuous symmetries and conserved quantities. Let a Lie group G act smoothly on $\mathcal{Q}$ by diffeomorphisms $\Phi_g : \mathcal{Q} \to \mathcal{Q}$. The corresponding action on a state $(q, \dot{q}) \in T\mathcal{Q}$ is its tangent lift,

$$T\Phi_g(q, \dot{q}) := \big(\Phi_g(q),\, D_q\Phi_g(\dot{q})\big),$$

where $D_q\Phi_g : T_q\mathcal{Q} \to T_{\Phi_g(q)}\mathcal{Q}$ is the *differential* of Φ_g at q. A Lagrangian is called G-*invariant* when

$$(L \circ T\Phi_g)(q, \dot{q}) = L(q, \dot{q}), \qquad g \in G. \tag{2}$$

Let $\mathfrak{g} = \mathfrak{g}(G)$ be the Lie algebra of G, and let $\xi_{\mathcal{Q}}(q) \in T_q\mathcal{Q}$ denote the infinitesimal generator associated with $\xi \in \mathfrak{g}$. Noether's theorem states that, along any Euler–Lagrange trajectory of a regular G-invariant Lagrangian,

$$J_\xi(q, \dot{q}) := \langle \partial_{\dot{q}} L(q, \dot{q}),\, \xi_{\mathcal{Q}}(q) \rangle \tag{3}$$

is constant in time. Here, $\partial_{\dot{q}} L(q, \dot{q}) \in T_q^*\mathcal{Q}$ is the fiber derivative, written in coordinates as the generalized momentum.

This identifies the architectural target used in the next section. We do not need the generalized coordinates themselves to transform linearly; we need the *learned scalar Lagrangian* to satisfy equation (2), and the lift and equivariant network make that invariance automatic for every value of the trainable parameters. The guarantee is a property of the continuous-time learned system: a generic numerical integrator may still introduce discretization drift in equation (3), even when the underlying learned flow conserves it exactly (Marsden and Ratiu, 1999).

3 Equivariant Lagrangian neural networks

The purpose of the architecture is to enforce the hypothesis of Noether's theorem rather than ask the optimizer to discover it from data. A conventional LNN may fit the observed trajectories while assigning different Lagrangian values to symmetry-related states, in which case the corresponding momentum is not protected by the model class. We construct L_θ so that equation (2) holds identically. The construction has two conceptual steps: first represent the physical state by features with known linear transformation laws, and then map those features to an invariant scalar.

3.1 From nonlinear generalized coordinates to linear features

Equivariant neural layers act on vector spaces carrying linear representations, but a symmetry need not act linearly on the generalized coordinates used by a mechanical model. Global rotations, for example, act nonlinearly on Euler angles, MRPs, and articulated joint coordinates. Hence, we linearize the *action on features*, not the coordinate chart itself.

For each system, we choose a finite-dimensional representation space V_x, a representation $\rho_x : G \to \mathrm{GL}(V_x)$, and a differentiable configuration feature map

$$\phi : \mathcal{Q} \to V_x$$

that obeys

$$\phi\big(\Phi_g(q)\big) = \rho_x(g)\phi(q), \qquad q \in \mathcal{Q},\ g \in G. \tag{4}$$

The maps used here are constructed from physical quantities whose transformation laws are known explicitly: planar angles by unit directions $(\cos\theta, \sin\theta)$, rigid-body orientations by attached frame axes, and point configurations by Cartesian positions. When a system defines m Cartesian position vectors $x_i(q) \in \mathbb{R}^d$, we collect the output of its position map as

$$X(q) := \big[x_1(q), \ldots, x_m(q)\big]^\top \in \mathbb{R}^{m \times d}. \tag{5}$$

Depending on the lift, the vectorization of $X(q)$ may form all or part of $\phi(q)$. It may also be retained as an auxiliary position map when a minimal lift does not include these Cartesian positions directly. The per-system choices are summarized in table 3. Thus the feature maps and their representations are chosen together so that the original action Φ becomes linear after lifting.

The same challenge is present for velocity. Networks will require a feature map $v : T\mathcal{Q} \to V_v$ with a linear transformation law

$$v(\Phi_g(q), D_q\Phi_g(\dot{q})) = \rho_v(g)v(q, \dot{q}),$$

for some representation $\rho_v : G \to \mathrm{GL}(V_v)$. One convenient way to construct such velocity features is to introduce a differentiable map $\kappa : \mathcal{Q} \to V_v$; its Jacobian $J_\kappa(q) := D_q\kappa$ is a linear map, and applying it to the velocity $\dot{q}$ gives the velocity feature

$$v(q, \dot{q}) = J_\kappa(q)\dot{q}.$$

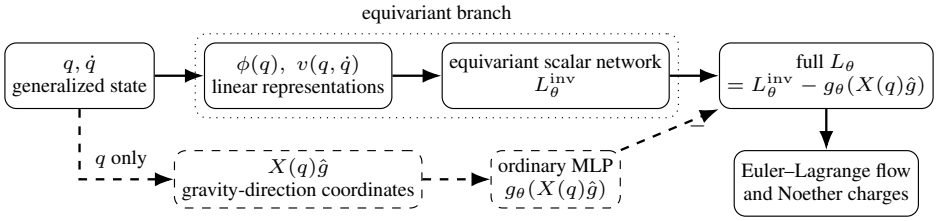

Figure 1: Architecture and gravity branch. The equivariant network maps the lifted state to L_θ^{inv}. A separate position-only branch projects the Cartesian positions $X(q)$ onto the fixed gravity direction $\hat{g}$ and subtracts the output of an ordinary MLP from the invariant branch.

When κ is equivariant, i.e. when $\kappa\big(\Phi_g(q)\big) = \rho_v(g)\kappa(q)$, then differentiating the equivariance identity gives the required transformation law for the velocity features:

$$J_\kappa\big(\Phi_g(q)\big) D_q\Phi_g(\dot{q}) = \rho_v(g) J_\kappa(q)\dot{q}.$$

To recap, $\phi(q)$ provides the configuration features used by the scalar model, while $J_\kappa(q)\dot{q}$ provides its velocity features; the two maps may differ when the physical symmetry requires it. The *lifted-state map* is

$$(q, \dot{q}) \mapsto (\phi(q), J_\kappa(q)\dot{q})\,,$$

and the output is the *lifted state*.

3.2 Invariant lifted Lagrangian and Noether guarantee

Once the lifted state transforms linearly, the remaining task is to make the scalar model invariant. Let

$$f_\theta : V_x \oplus V_v \to \mathbb{R}$$

satisfy

$$f_\theta\big(\rho_x(g)x, \rho_v(g)v\big) = f_\theta(x, v), \qquad g \in G. \tag{6}$$

We define the generalized-coordinate (learned) Lagrangian by

$$L_\theta(q, \dot{q}) = f_\theta(\phi(q), v(q, \dot{q})) + \varepsilon\langle v(q, \dot{q}), v(q, \dot{q})\rangle_H, \tag{7}$$

where $\varepsilon \geq 0$ is fixed and $\langle u, v\rangle_H = u^\top H v$ for a positive-definite matrix H. The second term is an optional kinetic floor used to improve numerical regularity at initialization. When it is used, we choose H to obey

$$\rho_v(g)^\top H \rho_v(g) = H, \qquad g \in G, \tag{8}$$

so that the stabilization does not break the symmetry; in all experiments we use $H = I$, admissible because the velocity features of every lift used transform orthogonally. When no floor is required, $\varepsilon = 0$.

Theorem 1 (Pullback invariance and learned Noether charges)**.** *Let G act smoothly on $\mathcal{Q}$ by Φ. Suppose ϕ and v satisfy the feature transformation laws above, f_θ satisfies equation (6), and, when $\varepsilon > 0$, H satisfies equation (8). Then the Lagrangian in equation (7) is G-invariant:*

$$(L_\theta \circ T\Phi_g)(q, \dot{q}) = L_\theta(q, \dot{q}) \qquad \textit{for every } g \in G.$$

If L_θ is regular, then for every $\xi \in \mathfrak{g} = \mathfrak{g}(G)$ the Euler–Lagrange flow conserves

$$J_{\theta,\xi}(q, \dot{q}) = \langle \partial_{\dot{q}} L_\theta(q, \dot{q}), \xi_{\mathcal{Q}}(q)\rangle\,.$$

The lift is responsible for converting the physical action into known linear transformation laws; the neural network is responsible for mapping those representations to an invariant scalar. The proof is given in appendix A.

3.3 Invariant scalar networks with factorized bilinear interactions

Theorem 1 leaves one neural design requirement: the lifted state must be mapped to an invariant scalar. Each hidden block in our implementation has the form equivariant linear map $\rightarrow$ factorized bilinear residual $\rightarrow$ gated nonlinearity, followed by an equivariant linear map to the trivial scalar representation.

An equivariant linear map is an intertwiner, parameterized with the generator-nullspace construction of Finzi et al. (2021): for every Lie algebra generator T_α, admissible weights satisfy $d\rho_{\text{out}}(T_\alpha)W - Wd\rho_{\text{in}}(T_\alpha) = 0$, and biases lie in the invariant subspace. Gated nonlinearities modulate nontrivial fields with invariant scalar gates; neither maps nor gates form contractions between vector-valued channels, so we add equivariant bilinear maps $B_p : V_{\text{L}} \times V_{\text{R}} \rightarrow V_{\text{O}}$ satisfying $B_p(\rho_{\text{L}}(g)u, \rho_{\text{R}}(g)w) = \rho_{\text{O}}(g)B_p(u, w)$, including invariant contractions such as dot products. We obtain the B_p by applying the same nullspace solver to the tensor-product generator action.

Suppose the hidden state contains m_{L} left channels $u_i \in V_{\text{L}}$, m_{R} right channels $w_j \in V_{\text{R}}$, and m_{O} output channels $z_s \in V_{\text{O}}$. A dense bilinear update would learn one coefficient per output, left, and right channel triple, i.e. $P \times m_{\text{O}} \times m_{\text{L}} \times m_{\text{R}}$ parameters; to avoid this cubic scaling, we parameterize the coefficients with rank r. The layer forms the mixtures $\bar{u}_{pk} = \sum_{i=1}^{m_{\text{L}}} A^{\text{L}}_{pki} u_i$ and $\bar{w}_{pk} = \sum_{j=1}^{m_{\text{R}}} A^{\text{R}}_{pkj} w_j$ from learned left, right, and output mixing factors, and evaluates the coefficients and the residual

$$C_{psij} = \sum_{k=1}^{r} A^{\text{O}}_{psk} A^{\text{L}}_{pki} A^{\text{R}}_{pkj}, \qquad \Delta z_s = \sum_{p=1}^{P} \sum_{k=1}^{r} A^{\text{O}}_{psk} B_p(\bar{u}_{pk}, \bar{w}_{pk}), \tag{9}$$

which is added to z_s, for a parameter count of $P \times r \times (m_{\text{L}} + m_{\text{R}} + m_{\text{O}})$. The learned factors act only on channel multiplicities, whereas the fixed maps B_p act on representation coordinates, so the rank restriction reduces expressivity without relaxing equivariance. Further details are given in appendix B.2.

3.4 Choosing the lift: analytic and minimal features

The conservation theorem requires the transformation laws above, but it does not select a unique feature map. We therefore use two levels of geometric information. An *analytic lift* may include the Cartesian position map $X(q)$ of equation (5), attachment offsets, and articulated frames derived from the system geometry. A *minimal lift* retains a smaller collection of features with the same required transformation laws, such as planar unit directions, spatial frame axes, or periodic joint phases. For gravity-enabled systems, both variants use the same auxiliary position map $X(q)$ to construct the gravity features, even when the minimal $\phi(q)$ does not contain those Cartesian positions. When a closed-form Jacobian–vector product is inconvenient, $J_\kappa(q)\dot{q}$ is evaluated by automatic differentiation.

A minimal lift can discard morphology that the analytic lift retains, and either map can be poorly conditioned in a particular chart; we compare analytic and minimal constructions empirically rather than folding their adequacy into theorem 1.

3.5 Translation symmetry

For systems with global translations, write $q = (c, r)$, where $c \in \mathbb{R}^d$ is absolute position and r contains translation-invariant coordinates. We omit c from the configuration lift, $\phi(c, r) = \phi(r)$, while the velocity features $v(q, \dot{q}) = J_\kappa(q)\dot{q}$ retain $\dot{c}$. The learned Lagrangian therefore has no explicit dependence on c, so c is cyclic and the conjugate momentum $p_c = \partial_{\dot{c}} L_\theta$ is conserved. We use this construction for Rotor, TwoBodySpring, and the centered Two-body lift, so both EQ-LNN and the lifted LNN inherit translation invariance from the map. For Two-body we therefore report P_x, P_y as lift-induced diagnostics and use L_z for the matched conservation comparison.

3.6 Gravity: a learned potential with residual symmetry

Let $\hat{g} \in \mathbb{R}^d$ be the fixed unit vector pointing in the gravity direction. Applying the position map from equation (5) gives the gravity features $X(q)\hat{g} = \left[x_1(q)^\top \hat{g}, \ldots, x_m(q)^\top \hat{g}\right]^\top \in \mathbb{R}^m$, which

contains the signed coordinate of every modeled position along the gravity direction. For systems without gravity, the invariant model in equation (7) is the full learned Lagrangian; for gravity-enabled systems, write that invariant branch as L_θ^{inv}. In the implementation (figure 1), an ordinary MLP learns a position-only scalar that is subtracted from this branch:

$$L_\theta(q, \dot{q}) = L_\theta^{\text{inv}}(q, \dot{q}) - g_\theta(X(q)\hat{g}) \,. \tag{10}$$

Thus the full Lagrangian uses both branches, but the gravity MLP consumes $X(q)\hat{g}$ rather than all of $\phi(q)$. The Cartesian positions in $X(q)$ may be part of an analytic lift or supplied by an auxiliary position map for a minimal lift.

No velocity feature enters this branch because uniform gravity is represented as a configuration-dependent potential. Its effect on acceleration arises only through the generalized force $-D_q[X(q)\hat{g}]^\top \nabla g_\theta(X(q)\hat{g})$, while the velocity Hessian $\partial^2_{\dot{q}\dot{q}} L_\theta$ is unchanged. Gravity is therefore learned rather than hard-coded.

Whenever the position map transforms as $X(\Phi_R(q)) = X(q)R^\top$, the gravity features are unchanged by rotations satisfying $R\hat{g} = \hat{g}$. Hence the full Lagrangian is invariant under the field stabilizer

$$G_{\hat{g}} := \{R \in G : R\hat{g} = \hat{g}\},$$

and the conclusion of theorem 1 applies to this residual group. A nonzero vertical field in three dimensions leaves an axial $\mathrm{SO}(2)$ symmetry; fixed gravity removes the continuous planar rotation symmetry of the pendulum systems, and for anchored systems the stabilizer can collapse further (appendix C): the implemented CoupledPendulum configuration retains no continuous symmetry under gravity.

4 Experiments

We seek to answer the following questions with our experiments: *(1) How does* EQ-LNN *compare with a dense LNN supplied with the same lifted inputs? (2) Do the learned dynamics conserve the Noether charges implied by the enforced symmetries, and only those charges? (3) Does equivariance improve generalization to symmetry-related states outside the training distribution? (4) How much system-specific structure is required in the lift?*

We study these questions across particle, articulated-body, and rigid-body systems in both planar and spatial settings: particle systems provide a controlled setting for translational and rotational symmetries, articulated systems introduce nonlinear kinematics, constraints, and external gravity, and rigid bodies additionally require modeling rotational inertia and angular momentum. Handling these structures is important for transitioning EQ-LNN towards applications in robotics and aerospace, where articulated and rigid-body dynamics interact with external forces, fields, and other environmental effects.

4.1 Systems, baselines, and protocol

We evaluate five primary systems with different mechanical structure. **ChainPendulum** is a planar serial pendulum, with one rotational degree of freedom per joint and each mass attached to the preceding link. **CoupledPendulum** instead consists of spatial pendula attached to fixed anchors by ball joints and coupled to one another by springs; each pendulum therefore has two angular degrees of freedom. Together with the **Gyroscope**, a rigid body with a fixed point under gravity, these systems follow the benchmarks of Finzi et al. (2020b). **Two-body** is the planar gravitational two-body problem, while **TwoBodySpring** replaces the gravitational interaction with a harmonic spring. Additional results for Rotor and the two Spacecraft systems are reported in appendix E; full system definitions and symmetry inventories are given in appendix C.

We compare EQ-LNN with LNN (Cranmer et al., 2020), HNN (Greydanus et al., 2019), and DeLaN (Lutter et al., 2019). We also use a *lifted LNN* that receives the same lifted state as EQ-LNN but replaces the equivariant network with an ordinary dense MLP. This provides a matched comparison between using the lift alone and additionally enforcing equivariance. Together these models form a supervision ladder from raw coordinates, through the shared lift, to the invariant scalar network; table 9 adds the Cartesian-constraint family.

Table 1: Median trajectory MAE over five seeds. Lifted LNN uses the same lift as EQ-LNN; Gain is lifted-LNN/EQ-LNN, so values above one favor EQ-LNN. Standard errors are in appendix E.1; HNN and DeLaN are non-matched comparisons.

System	EQ-LNN	Lifted LNN	Raw LNN	HNN	DeLaN	Gain
Two-body	$\mathbf{1.77\times10^{-2}}$	4.26×10^{-2}	3.79×10^{-1}	6.86×10^{-2}	6.95×10^{-2}	$2.4\times$
TwoBodySpring	$\mathbf{7.90\times10^{-5}}$	4.14×10^{-3}	6.58×10^{-2}	3.51×10^{-2}	3.49×10^{-2}	$52.5\times$
ChainPendulum	$\mathbf{9.02\times10^{-4}}$	1.64×10^{-2}	4.62×10^{-1}	1.02×10^{-1}	9.95×10^{-2}	$18.2\times$
CoupledPendulum	4.68×10^{-2}	$\mathbf{3.91\times10^{-2}}$	1.60×10^{-1}	1.51×10^{-1}	1.50×10^{-1}	$0.84\times$
Gyroscope	$\mathbf{4.95\times10^{-3}}$	1.92×10^{-1}	3.01×10^{-1}	6.27×10^{-2}	6.00×10^{-2}	$38.8\times$

Unless noted otherwise, models within each matched comparison use the same training trajectories and optimization settings. We report median trajectory MAE over seeds, the mean absolute full-state rollout error in the physical embedding (appendix C.4). All main-table EQ-LNN and lifted-LNN cells use minimal lifts (section 3.4); table 2 compares the analytic variants, which coincide with the minimal maps on TwoBodySpring and CoupledPendulum. Training details are given in appendices C.2 and C.3, and seed-level statistics in appendix E.1.

4.2 Predictive performance with matched lifted inputs

We first compare EQ-LNN with standard learned-dynamics baselines. Across the five primary systems, EQ-LNN achieves lower median trajectory error than the raw-coordinate LNN, HNN, and DeLaN (table 1). The improvement holds across particle, articulated, and rigid-body systems.

As shown in figure 2, the lift accounts for some of the improvement over the raw-coordinate models, but EQ-LNN further improves on the matched lifted LNN for four of the five systems. CoupledPendulum is the exception, where the lifted LNN performs slightly better. Because the lift is shared, these gains cannot be explained by the lifted inputs alone; they support the complete EQ-LNN architecture rather than any individual equivariant component.

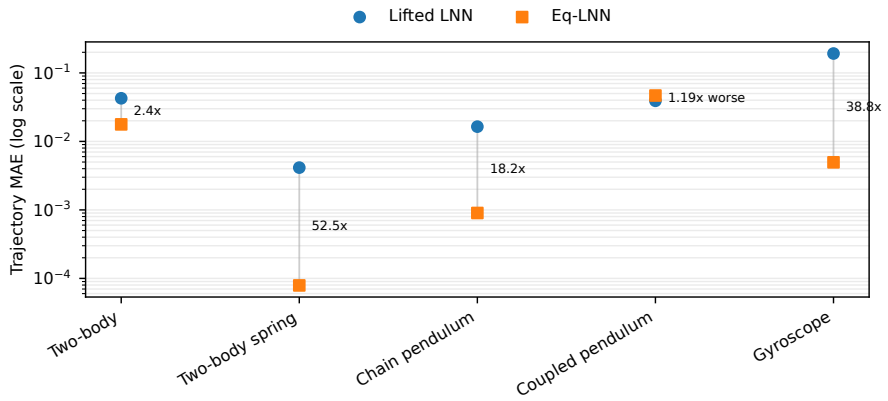

Figure 2: Matched comparison using identical lifts and five seeds. Markers show system medians; labels give the lifted-LNN/EQ-LNN trajectory-MAE ratio.

4.3 Noether charge conservation

We next test whether the symmetries enforced by EQ-LNN produce the conserved quantities predicted by theorem 1. For each enforced generator, we compute the corresponding learned Noether charge along trajectories generated by the learned dynamics and measure its normalized maximum drift. The primary systems with enforced continuous symmetries are Two-body (L_z), TwoBodySpring (P_x, P_y, L_z), and Gyroscope (L_z) (appendix E).

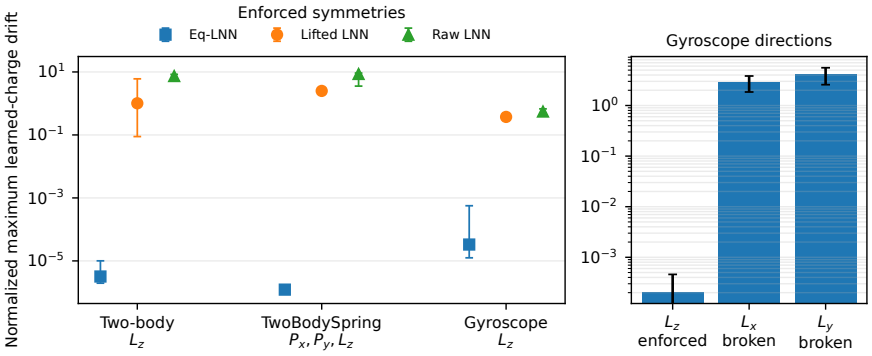

Figure 3: Learned Noether-charge drift along model rollouts. **Left:** maximum normalized drift over charges enforced by each architecture. **Right:** the Gyroscope EQ-LNN conserves the enforced gravity-axis charge L_z, while the broken L_x and L_y components drift. Markers show medians over seeds and bars span seeds. Additional per-symmetry results, including Rotor, are given in appendix E.

Table 2: Median trajectory MAE for analytic and minimal EQ-LNN lifts over five seeds. A dagger marks identical maps. No analytic Gyroscope model was trained; Rotor, Spacecraft, and per-seed results are in appendix E.

System	Analytic EQ-LNN	Minimal-lift EQ-LNN
Two-body	$\mathbf{1.58\times10^{-2}}$	1.77×10^{-2}
TwoBodySpring	7.90×10^{-5}	7.90×10^{-5} †
ChainPendulum	$\mathbf{7.83\times10^{-4}}$	9.02×10^{-4}
CoupledPendulum	4.68×10^{-2}	4.68×10^{-2} †

As shown in figure 3, EQ-LNN exhibits small numerical drift in the enforced charges under finite-step integration, while the lifted and raw LNNs exhibit substantially larger drift. The Gyroscope provides a useful within-model control: gravity preserves rotations about the vertical axis but breaks rotations about the other two axes, and the same EQ-LNN accordingly conserves L_z while L_x and L_y drift.

Across the 45 evaluated checkpoints, the EQ-LNN value is conserved to numerical tolerance, with normalized drift between 9.0×10^{-11} and 5.7×10^{-4} across seeds. Matched lifted LNNs show order-one drift on the same rotational charges, a gap of three to six orders of magnitude.

The enforced symmetries should also transfer to symmetry-related states outside the training distribution. To test this, we train on TwoBodySpring trajectories drawn from a ± 0.25-rad orientation sector and evaluate on states rotated by $\pi/2$ out of the sector; the test differs from training only by the enforced rotation. The EQ-LNN integrated log relative error is unchanged to the reported precision, the matched lifted LNN degrades, and the raw LNN is nearly unchanged. The full plot is in appendix E.3.2.

4.4 COMPARING LIFT METHODOLOGIES

The main-table EQ-LNN and lifted-LNN cells all use minimal lifts, so the matched gains reported there already reflect the low-design setting (table 1 and appendix E.1).

Minimal lifts nearly match analytic lifts on Two-body and ChainPendulum and are identical by construction on CoupledPendulum and TwoBodySpring. SpacecraftA is the counterexample: removing attachment offsets and appendage frames raises MAE from 3.33×10^{-4} (analytic arm under the earlier protocol) to 6.28×10^{-2}, a $188\times$ gap (figure 4). Equivariance cannot recover geometry discarded by the lift.

5 Related work

HNNs learn Hamiltonians and derive canonical vector fields; SymODEN and Hamiltonian graph networks extend this idea to controlled and relational systems (Greydanus et al., 2019; Zhong et al., 2020; Sanchez-Gonzalez et al., 2019). LNNs instead learn a full Lagrangian without requiring canonical momenta, while DeLaN restricts it to a learned positive-definite mass matrix and potential (Cranmer et al., 2020; Lutter et al., 2019). These models encode energy or variational structure, but do not generally impose prescribed spatial group actions and their associated momentum maps.

Müller (2023) uses Noether symmetry to construct neural integrators with exact conservation laws, while MCLNN enforces Euclidean symmetries in particle systems through relational features (Bhattoo et al., 2022b). Other methods discover symmetries or invariants from data using discrete Lagrangians, meta-learned conservation losses, or direct conservation-law discovery (Lishkova et al., 2022; Alet et al., 2021; Liu and Tegmark, 2022). Vaquero et al. (2024) learns symmetry-preserving Hamiltonian maps through invariant Lagrangian submanifolds. EQ-LNN instead assumes a known symmetry and targets nonlinear actions on generalized coordinates, lifted to linear representations before constructing an invariant full Lagrangian for particle, articulated, and rigid-body systems with residual symmetries under fixed gravity.

Group-equivariant networks encode symmetry through architectural weight sharing (Cohen and Welling, 2016); EMLP constructs equivariant maps for arbitrary matrix groups (Finzi et al., 2021), while tensor-field, attention, and graph architectures specialize this principle to Euclidean data (Thomas et al., 2018; Fuchs et al., 2020; Finzi et al., 2020a; Satorras et al., 2021). GMN combines generalized coordinates, forward kinematics, and equivariant graph prediction (Huang et al., 2022), while Lagrangian graph networks model articulated topology and size generalization (Bhattoo et al., 2022a). EQ-LNN differs by making the equivariant network output a scalar Lagrangian, so invariance yields learned Noether charges directly.

CHNN and CLNN evolve redundant Cartesian states and enforce known constraints through Lagrange multipliers (Finzi et al., 2020b). They provide stronger mechanistic supervision on these benchmarks; EQ-LNN instead retains reduced coordinates and supplies a group action plus a symmetry-compatible lift, trading exact constraints for a broader invariant Lagrangian.

6 Conclusion and limitations

We introduced EQ-LNN, which enforces physical symmetries in a learned Lagrangian while retaining nonlinear generalized coordinates. Symmetry-compatible configuration and velocity features allow the same construction to handle particle, articulated, and rigid-body systems, while translations and fixed external fields are incorporated through the symmetries that remain. In our experiments, the learned dynamics conserve the momenta associated with enforced symmetries, while symmetry-broken quantities are free to change. EQ-LNN improves trajectory prediction over the matched lifted LNN on four of five primary systems, and enforced symmetries transfer to rotated states outside the training sector. Known symmetries can thus be built directly into learned Lagrangian dynamics without giving up compact generalized-coordinate models.

Our approach has several limitations: the Noether guarantees hold for the regular continuous-time learned dynamics, and numerical integration still introduces small charge drift. The implementation targets connected continuous symmetries; disconnected transformations require additional constraints. Equivariance also does not guarantee better prediction (CoupledPendulum), and it cannot recover geometry discarded by the lift (figure 4). The learned-charge evaluation excludes the spacecraft checkpoints. Finally, we have not compared the factorized bilinear layer against a dense tensor-product implementation.

Reproducibility statement

The appendix provides proofs, the architecture and bilinear-layer specification, system definitions and protocols, per-system lift formulations, seed-variability summaries, learned-charge metrics, coordinate diagnostics, and the method-defining code listings (appendix F). The full experiment codebase, which incorporates the benchmark reference implementation, cannot be released in full;

the method-defining code listings (appendix F) together with the training and evaluation protocols (appendix C) specify the implemented procedures.

ETHICS STATEMENT

This study uses synthetic mechanical systems and no human data. For claims that mix theoretical guarantees with empirical measurements, we state explicitly which quantities hold by construction, which are measured along the learned flow, and which are subject to numerical error.

AI USE STATEMENT

Generative-AI tools were used iteratively during software development and manuscript preparation. The codebase combines author-written code with a pre-existing benchmark implementation; AI assisted with integration, debugging, and extensions, particularly symmetry-compatible lifts, their Jacobians, and related equivariant-model components. AI-assisted system implementation was used for TwoBodySpring and the two Spacecraft systems, with the latter completed and debugged from incomplete pre-existing implementations. AI also assisted with mathematical formalization and the proof of Theorem 1, and with iterative drafting, revision, and proofreading of the manuscript. The authors inspected and tested the generated code, vetted the mathematical arguments and numerical claims, made the final decisions on analysis and presentation, and take responsibility for all content in the paper.

REFERENCES

Ferran Alet, Dylan Doblar, Allan Zhou, Joshua B. Tenenbaum, Kenji Kawaguchi, and Chelsea Finn. Noether networks: Meta-learning useful conserved quantities. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Ravinder Bhattoo, Sayan Ranu, and N. M. Anoop Krishnan. Learning articulated rigid body dynamics with lagrangian graph neural network. In *Advances in Neural Information Processing Systems*, 2022a.

Ravinder Bhattoo, Sayan Ranu, and N. M. Anoop Krishnan. Lagrangian neural network with differentiable symmetries and relational inductive bias. In *International Conference on Learning Representations*, 2022b.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, 2016.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *Proceedings of the 37th International Conference on Machine Learning*, 2020a.

Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. In *Advances in Neural Information Processing Systems*, 2020b.

Marc Finzi, Max Welling, and Andrew Gordon Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, and Max Welling. SE(3)-transformers: 3d roto-translation equivariant attention networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Wenbing Huang, Jiaqi Han, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Equivariant graph mechanics networks with constraints. In *International Conference on Learning Representations*, 2022.

Yana Lishkova, Paul Scherer, Steffen Ridderbusch, Mateja Jamnik, Pietro Liò, Sina Ober-Blöbaum, and Christian Offen. Discrete lagrangian neural networks with automatic symmetry discovery. *arXiv preprint arXiv:2211.10830*, 2022.

Ziming Liu and Max Tegmark. AI Poincaré: Machine-learning conservation laws from trajectories. *Physical Review D*, 106(4):045011, 2022.

Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*, 2019.

Jerrold E. Marsden and Tudor S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer, 2 edition, 1999.

Eike Hermann Müller. Exact conservation laws for neural network integrators of dynamical systems. *Journal of Computational Physics*, 488:112234, 2023. doi: 10.1016/j.jcp.2023.112234.

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ODE integrators. *arXiv preprint arXiv:1909.12790*, 2019.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhase, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Miguel Vaquero, Jorge Cortés, and David Martín de Diego. Symmetry preservation in hamiltonian systems: Simulation and learning. *Journal of Nonlinear Science*, 34(6):115, 2024. doi: 10.1007/s00332-024-10089-5.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020.

A Proof of the pullback theorem

Proof of theorem 1. Fix $g \in G$ and $(q, \dot{q}) \in T\mathcal{Q}$. By the assumed transformation laws of the lifted features,

$$\phi\big(\Phi_g(q)\big) = \rho_x(g)\phi(q), \qquad v(\Phi_g(q), D_q\Phi_g(\dot{q})) = \rho_v(g)v(q, \dot{q}).$$

Applying equation (6) to the learned term in equation (7) gives

$$f_\theta(\phi(\Phi_g(q)), v(\Phi_g(q), D_q\Phi_g(\dot{q}))) = f_\theta(\phi(q), v(q, \dot{q}))\,.$$

If $\varepsilon > 0$, equation (8) similarly implies

$$\langle \rho_v(g)v, \rho_v(g)v\rangle_H = v^\top \rho_v(g)^\top H \rho_v(g) v = \langle v, v\rangle_H.$$

Together, this gives

$$L_\theta(\Phi_g(q), D_q\Phi_g(\dot{q})) = L_\theta(q, \dot{q}),$$

so L_θ is G-invariant. If L_θ is regular, its Euler–Lagrange equations define a local flow. Noether's theorem then gives, for every $\xi \in \mathfrak{g}(G)$,

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\langle \partial_{\dot{q}} L_\theta(q, \dot{q}), \xi_{\mathcal{Q}}(q)\rangle = 0.$$

□

A.1 Residual invariance of the gravity branch

When gravity is present, denote the G-invariant Lagrangian established above by L_θ^{inv} and form the full model as in equation (10). Suppose $G \subseteq \mathrm{O}(d)$ acts on the landmark rows by

$$X\big(\Phi_R(q)\big) = X(q)R^\top.$$

For any $R \in G_{\hat{g}}$, orthogonality and $R\hat{g} = \hat{g}$ imply $R^\top \hat{g} = \hat{g}$, and therefore

$$X\big(\Phi_R(q)\big)\hat{g} = X(q)R^\top \hat{g} = X(q)\hat{g}.$$

Thus $g_\theta(X(q)\hat{g})$ is invariant under the residual group $G_{\hat{g}}$ even though g_θ is an unconstrained MLP. Since L_θ^{inv} is G-invariant, the full Lagrangian

$$L_\theta = L_\theta^{\mathrm{inv}} - g_\theta(X(q)\hat{g})$$

is $G_{\hat{g}}$-invariant, and the Noether conclusion of theorem 1 applies to every generator in $\mathfrak{g}(G_{\hat{g}})$. When the system is anchored to the environment, $G_{\hat{g}}$ can be smaller than the free-space group and may even be trivial, in which case no continuous Noether guarantee remains; appendix C records the surviving symmetries for each system.

Moreover, the gravity term is independent of $\dot{q}$, so

$$\partial_{\dot{q}} L_\theta = \partial_{\dot{q}} L_\theta^{\mathrm{inv}}, \qquad \partial^2_{\dot{q}\dot{q}} L_\theta = \partial^2_{\dot{q}\dot{q}} L_\theta^{\mathrm{inv}}.$$

It changes the generalized-force term through $\partial_q L_\theta$, but does not directly change the learned momentum or velocity Hessian.

B Architecture details

B.1 Intertwiner and invariant bases

For generator matrices T_a^{in} and T_a^{out}, vectorizing the intertwiner equation produces the linear constraints

$$\big(T_a^{\mathrm{out}} \otimes I - I \otimes (T_a^{\mathrm{in}})^\top\big)\,\mathrm{vec}(W) = 0.$$

An SVD nullspace basis is computed once for each input/output fiber-type pair; trainable coefficients mix these bases across representation multiplicities. Bias bases are obtained from $T_a b = 0$.

B.2 Factorized bilinear interactions

For one representation-type path $V_{\mathrm{L}} \otimes V_{\mathrm{R}} \to V_{\mathrm{O}}$, the infinitesimal tensor-product action is

$$T_\alpha^{\mathrm{L}\otimes\mathrm{R}} = T_\alpha^{\mathrm{L}} \otimes I + I \otimes T_\alpha^{\mathrm{R}}.$$

Applying the linear intertwiner solver to this representation yields a basis $B_p \in \mathbb{R}^{d_{\mathrm{O}}\times d_{\mathrm{L}}\times d_{\mathrm{R}}}$ of equivariant bilinear maps. After identifying each bilinear map with a linear map on $V_{\mathrm{L}} \otimes V_{\mathrm{R}}$, their span is

$$\mathrm{Hom}_G(V_{\mathrm{L}} \otimes V_{\mathrm{R}}, V_{\mathrm{O}}) := \left\{ B : V_{\mathrm{L}} \otimes V_{\mathrm{R}} \to V_{\mathrm{O}} \;\middle|\; B\big(\rho_{\mathrm{L}}(g) \otimes \rho_{\mathrm{R}}(g)\big) = \rho_{\mathrm{O}}(g)B,\ \forall g \in G \right\}.$$

Here ρ_{L}, ρ_{R}, and ρ_{O} denote the corresponding group actions: this is the vector space of linear maps that commute with the action of G. The basis is computed once and stored as a non-trainable buffer. It is the numerical analogue of a Clebsch–Gordan coupling basis for the supplied matrix representations. Because the current solve uses infinitesimal generators, it enforces the connected component generated by them; disconnected components require additional finite constraints.

A dense layer would learn one coefficient for every basis map and left, right, and output multiplicity triple—the $P \times m_{\mathrm{O}} \times m_{\mathrm{L}} \times m_{\mathrm{R}}$ coefficients of equation (9). The implemented layer instead factorizes that multiplicity tensor with rank r,

$$C_{psij} = \sum_{k=1}^{r} A^{\mathrm{O}}_{psk} A^{\mathrm{L}}_{pki} A^{\mathrm{R}}_{pkj},$$

which gives the staged computation in equation (9). This is a CP factorization of the multiplicity tensor only: the geometric basis B_p is not approximated. Consequently, decreasing r restricts channel mixing but preserves equivariance exactly up to numerical nullspace and floating-point error. The reported default is $r = 8$; for an $\mathrm{SO}(2)$ chain model with width 256 and three layers, the factorized blocks contribute 16,320 parameters, versus 7,369,500 for a dense coefficient tensor over the same multiplicities.

The bilinear block uses nontrivial fields as inputs. It sends all allowed invariant contractions to scalar channels and, when the corresponding basis is nonempty, sends field-valued products to field types already present in the hidden representation. Scalar–scalar nonlinearities and scalar–field products are handled by scalar activations and gates.

B.3 Per-system instantiation

The construction is fully determined only once each system's fiber types are fixed. Field multiplicities are set by each lift's specification—repeated scalar and vector fields per landmark, with input and output types following the lift—and are not a free hyperparameter. The network is otherwise identical across systems (width 256, three hidden layers, gated nonlinearity, bilinear rank $r = 8$). Every system uses the kinetic floor of equation (7) with $\varepsilon = 3 \times 10^{-2}$ and $H = I$, i.e. $\varepsilon\|J_\kappa(q)\dot{q}\|^2$; the velocity features of every lift used transform orthogonally, so equation (8) holds. table 3 records the per-system chart and primary lift, and appendix D gives the corresponding map definitions.

B.4 Finite-precision verification

The test suite checks linear and bilinear equivariance, nonzero field-to-scalar gradients, invariant scalar outputs, analytic Jacobians against autodiff, tangent equivariance under nonlinear coordinate changes, translation nullspaces, random-parameter shadow-switch invariance, and coordinate-invariant scoring. These tests verify architectural identities independently of training. The learned-charge module adds eight targeted tests covering identity and nonlinear coordinate-change pushforwards, agreement with physical momentum maps, conservation for a symmetric reference Lagrangian, and autodiff/finite-difference consistency. Reference implementations of the intertwiner bases, equivariant layers, lifted nets, and minimal lift are given in appendix F.

C Systems, symmetries, and maps

Enforced symmetries. The learned-charge evaluation counts a symmetry as enforced only when the model and map construction represents that action, and names each symmetry by the momentum

Table 3: Benchmark inventory. E, P, and L denote energy, linear momentum, and angular momentum of the true system, and COM denotes the center of mass. Under gravity, the listed spatial charge is the surviving one. Analytic and minimal-lift map definitions are collected in appendix D.

System	Chart	Exact spatial symmetry	True charges	Primary map
ChainPendulum	joint angles	none after gravity	E	analytic / minimal (angles)
CoupledPendulum	two angles	none after gravity	E	analytic / unit directions
Gyroscope	ZXZ	rotations about gravity	E, L_z	attached frame / frame rows
Rotor	COM + MRP	$\mathrm{SO}(3)$ + translations	E, L, P	frame rows
SpacecraftB	COM + MRP	$\mathrm{SO}(3)$ + translations	E, L, P	frame rows
SpacecraftA	MRP + 2 phases	$\mathrm{SO}(3)$ + translations	E, L, P	landmarks or frame+phases
Two-body	Cartesian	$\mathrm{SO}(2)$ + translations	E, L_z, P	centered points
TwoBodySpring	Cartesian	$\mathrm{SO}(2)$ + translations	E, L_z, P	centered points

component it conserves. Rotor enforces all three translations and all three rotations; TwoBodySpring enforces P_x, P_y, L_z; Gyroscope enforces only L_z after fixing gravity; and the Two-body configuration enforces L_z, with P_x, P_y reported as lift-induced diagnostics: the centered lift makes both lifted architectures translation-invariant. The table above lists symmetries of the true system, which can therefore be broader than the subset enforced in a particular model. The evaluation covers Rotor, Two-body, TwoBodySpring, and Gyroscope: the planar pendula have no enforced symmetry under fixed gravity, and the spacecraft checkpoints were not included in the final evaluation batch. For the pendula this is not a limitation of the kinetic construction, which remains equivariant; the fixed gravity projection breaks every candidate rotation, so the full Lagrangian, and therefore the enforced set, is trivial there.

Construction identities. SpacecraftB analytic and minimal-lift EQ-LNN share the same map and are bit-identical at equal seeds. The same is true for TwoBodySpring. These cells verify common plumbing but are not counted as independent evidence for minimal-lift sufficiency. On default CoupledPendulum, the analytic and unit-length minimal maps differ only by a fixed positive scale, so the unequal-length variant is the actual coarsening experiment.

System provenance. The ChainPendulum, CoupledPendulum, Gyroscope, and Rotor systems follow the benchmark of Finzi et al. (2020b); their geometry and physical construction use the reference implementation.[1] The exact sampling statement for these systems is the `sample_initialConditions` implementation.

C.1 Data generation

One "system" is one initial-condition draw, i.e. one trajectory; physical parameters are class constants and are never sampled. Each matched study uses noiseless generated trajectories pooled per split mode: the pool cache is keyed by the body, pool size, and split mode, and the pool seed is the base seed plus the split-mode offset (0, 1, 2 for train/validation/test), so validation and test come from pools separate from training; every arm of a study consumes the identical pools. Each training trajectory contributes one random five-step chunk per epoch. Ground truth for every system integrates in float64 with dopri5 at tolerance 10^{-7} on the body grid of table 5; all data are noiseless.

C.2 Training protocol

Training minimizes the same trajectory-MAE function used for evaluation, applied to five-step chunks. The chunk rollout shares the fixed-step RK4 machinery of the eval protocol, so training and evaluation use a single rollout path. Each benchmark family samples 10,000 random initial conditions from its fixed physical parameters and integrates them into noiseless ground-truth trajectories; training uses 800 of these trajectories, with 100 validation and 100 test trajectories drawn from separate pools. One epoch streams the 800 training trajectories as four batches of 200, so a 500-epoch run is 2,000 Adam steps at a constant per-system learning rate (appendix C.3); there is no schedule, and gradients are clipped at norm 1. Checkpoints are taken at the final epoch; no best-validation selection is used.

[1] https://github.com/mfinzi/constrained-hamiltonian-neural-networks

Linear layers use orthogonal initialization, lifted-LNN heads zero-init their final layer, and intertwiner coefficients draw with $1/\sqrt{\text{numel}}$ scaling. The seed controls initialization, chunk selection, and batch order only; the data pools are shared across all arms. Every arm—baselines included—runs the identical training and evaluation path at its system's learning rate, with no per-baseline tuning beyond the recorded sweep. Training runs in float32; ground-truth integration and the learned-charge diagnostics run in float64. algorithm 1 summarizes construction and training; algorithm 2 below fixes the shared evaluation.

Algorithm 1 EQ-LNN construction and training

Require: trajectories $\{(q_i(t), \dot{q}_i(t))\}$; maps (ϕ, κ); invariant network f_θ; invariant metric H; weight ϵ; chunk length; epochs; learning rate

lift: $x = \phi(q), v = J_\kappa(q)\,\dot{q}$ ▷ J_κ analytic when available, otherwise autodiff
Lagrangian: $L_\theta(q, \dot{q}) = f_\theta(x, v) + \epsilon\,\|v\|_H^2$ ▷ equation (7)
vector field: solve $\frac{d}{dt}\partial_{\dot{q}} L_\theta - \partial_q L_\theta = 0$ for $\ddot{q}$ by autodiff
for each epoch **do**
    sample a batch of trajectories and start times
    integrate the learned flow for one multi-step chunk from each initial state
    loss ← trajectory MAE between predicted and true $(q, \dot{q})$ over the chunk
    Adam update of θ
end for
return trained L_θ; rollouts reuse the same lift and vector field

C.3 LEARNING RATES

Learning rates are set per system and are uniform across all arms of a system. The values in table 4 apply to every cell of the results tables.

Table 4: Learning rates per system.

System	Learning rate	Arms
ChainPendulum	2×10^{-3}	all arms
CoupledPendulum	3×10^{-3}	all arms
Gyroscope	3×10^{-3}	all arms
Two-body	1×10^{-3}	all arms
TwoBodySpring	1×10^{-3}	all arms
Rotor	3×10^{-3}	all arms
SpacecraftA	3×10^{-3}	all arms
SpacecraftB	3×10^{-3}	all arms

C.4 TRAJECTORY-MAE EVALUATION PROTOCOL

Trajectory MAE is computed by a single shared function (`DynamicsModel.trajectory_mae`) for the training loss, validation, and every results table. It evaluates the *full rollout state*—generalized coordinates and velocities, contributing equally—as the raw elementwise mean $\text{mean}\,|\text{pred} - \text{true}|$ over grid points and state dimensions, with no normalization. Chart-invariant bodies (Rotor, SpacecraftA/B) score on the differentiable physical embedding; Euclidean bodies score raw Cartesian states. algorithm 2 fixes the per-run protocol: take the median over test trajectories, then the median over seeds. A rollout that remains non-finite after the five-fold substep retry is scored by the zero-rollout fallback and labeled `run_status` $\in \{$`zero_fallback`, `partial_zero_fallback`$\}$; no redo trial triggered this fallback. Training-time divergence is stopped by a consecutive-skip breaker (50 skipped batches) and the trial is retained as failed.

Algorithm 2 Trajectory-MAE evaluation (shared, per run)

1: load checkpoint; for each test system: integrate the learned flow from its ICs on the eval grid (dt, T_{eval}) of table 5 (fixed-step RK4; five-fold substep retry on non-finite states)
2: score the full rollout state $(q, \dot{q})$ in raw chart units; chart-invariant bodies score on the physical embedding (table 3)
3: if the rollout remains non-finite: score with the zero-rollout fallback value and set `run_status`
4: error $\leftarrow \text{mean}_{t,k} \,|\, z_k^{\text{pred}}(t) - z_k^{\text{true}}(t) \,|$; median over test trajectories

Table 5: Per-system evaluation settings. Model rollouts use fixed-step RK4 on this grid; ground truth uses float64 dopri5 at the listed tolerance.

System	dt	T_{eval} (s)	Steps	Solver tolerance
ChainPendulum	0.03	3	100	10^{-7}
CoupledPendulum	0.05	5	100	10^{-7}
Gyroscope	0.02	2	100	10^{-7}
Rotor	0.05	5	100	10^{-7}
SpacecraftA	0.02	4	200	10^{-7}
SpacecraftB	0.02	4	200	10^{-7}
Two-body	0.4	20	50	10^{-7}
TwoBodySpring	0.05	10	200	10^{-7}

D System and Lift Formulations

This appendix gives the dynamics of the TwoBodySpring system introduced in this work and records the analytic and minimal lifts used in the experiments. The dynamics of the remaining benchmark systems follow the original benchmark formulations and are not repeated here.

D.1 Lift convention

The standard construction uses one configuration map $\phi : Q \to V$ and its tangent map,

$$(q, \dot{q}) \longmapsto \big(\phi(q), J_\phi(q)\dot{q}\big), \qquad J_\phi(q) = D\phi(q). \tag{11}$$

This is the form used by the anchored lifts, analytic and minimal alike. For free systems, a minor implementation refinement removes global position without removing the uniform-translation velocity mode:

$$(q, \dot{q}) \longmapsto \big(\phi_c(q), J_\kappa(q)\dot{q}\big). \tag{12}$$

Here ϕ_c is centered or relative, while κ contains the corresponding absolute landmarks. Thus translations are absent from the configuration features, but center-of-mass kinetic energy remains representable. equation (11) is recovered when $\phi_c = \kappa = \phi$. Constant anchor offsets, as in the coupled pendulum, do not affect the Jacobian and are equivalent at the tangent level.

For systems with uniform gravity, we additionally evaluate the Cartesian position map $X(q) \in \mathbb{R}^{m\times d}$ defined in equation (5). We keep this map conceptually distinct from ϕ and κ: the invariant branch receives $\big(\phi(q), J_\kappa(q)\dot{q}\big)$, whereas the gravity branch receives only $X(q)\hat{g}$, with $\hat{g}$ the fixed gravity direction. When the analytic lift already consists of these landmarks, a fixed vectorization of $X(q)$ is part of $\phi(q)$ and/or $\kappa(q)$. For a minimal lift, the same $X(q)$ is retained as an auxiliary analytic readout even when its landmarks are absent from $\phi(q)$. No velocity is passed to the gravity MLP; the derivative of this position-only term enters through $\partial_q L_\theta$ in the Euler–Lagrange equations.

For n points in $\mathbb{R}^d$, define the centering matrix

$$C_{n,d} = \left(\mathrm{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right) \otimes \mathrm{I}_d. \tag{13}$$

All tangent features below are obtained by differentiating the stated kinematic map. The implementation evaluates the minimal-lift Jacobians either in closed form or by forward-mode automatic differentiation; these are two implementations of the same map, not different models.

D.2 TWO-BODY HARMONIC SPRING

Two equal masses m move freely in $\mathbb{R}^2$ and interact through a zero-rest-length harmonic spring. With $q = (x_1^\top, x_2^\top)^\top \in \mathbb{R}^4$, the Lagrangian is

$$L(q, \dot{q}) = \frac{m}{2}\big(\|\dot{x}_1\|_2^2 + \|\dot{x}_2\|_2^2\big) - \frac{k}{2}\|x_2 - x_1\|_2^2. \tag{14}$$

The equations of motion are

$$m\ddot{x}_1 = k(x_2 - x_1), \qquad m\ddot{x}_2 = -k(x_2 - x_1). \tag{15}$$

Equivalently, for $c = (x_1 + x_2)/2$ and $r = x_2 - x_1$,

$$\ddot{c} = 0, \qquad \ddot{r} = -\frac{2k}{m} r. \tag{16}$$

The $\mathrm{SE}(2)$ symmetry yields conserved total linear and angular momentum,

$$P = m(\dot{x}_1 + \dot{x}_2), \qquad L_z = m \sum_{i=1}^{2} x_i \times \dot{x}_i, \tag{17}$$

where $a \times b = a_1 b_2 - a_2 b_1$.

Both reported lift variants use the same centered configuration and absolute kinematics,

$$\phi_c(q) = C_{2,2} q, \qquad \kappa(q) = q, \tag{18}$$

so $J_{\phi_c} = C_{2,2}$ and $J_\kappa = \mathrm{I}_4$; the analytic and minimal lifts coincide on this system.

D.3 COORDINATE PRIMITIVES

The remaining lifts are assembled from a small set of maps. For a planar angle, define

$$h(\theta) = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \qquad Dh(\theta) = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}. \tag{19}$$

For spherical coordinates (φ, ϑ), define

$$v_\ell(\varphi, \vartheta) = \ell \begin{bmatrix} -\cos\vartheta \\ \sin\vartheta \sin\varphi \\ \sin\vartheta \cos\varphi \end{bmatrix}. \tag{20}$$

Its two Jacobian columns are

$$\partial_\varphi v_\ell = \ell \begin{bmatrix} 0 \\ \sin\vartheta \cos\varphi \\ -\sin\vartheta \sin\varphi \end{bmatrix}, \qquad \partial_\vartheta v_\ell = \ell \begin{bmatrix} \sin\vartheta \\ \cos\vartheta \sin\varphi \\ \cos\vartheta \cos\varphi \end{bmatrix}. \tag{21}$$

For intrinsic ZXZ angles $\eta = (\alpha, \beta, \gamma)$, let

$$F_{\mathrm{ZXZ}}(\eta) = \big(R_z(\alpha) R_x(\beta) R_z(\gamma)\big)^\top, \tag{22}$$

and denote its rows by f_1, f_2, f_3. For modified Rodrigues parameters $\sigma \in \mathbb{R}^3$, the free-body frame is

$$F_{\mathrm{MRP}}(\sigma) = \mathrm{I}_3 + \frac{8\,[\sigma]_\times^2 - 4(1 - \|\sigma\|_2^2)\,[\sigma]_\times}{(1 + \|\sigma\|_2^2)^2}, \tag{23}$$

again with rows f_1, f_2, f_3. The frame Jacobians are obtained by analytically differentiating these expressions. The MRP shadow transform $\sigma^{\mathrm{S}} = -\sigma/\|\sigma\|_2^2$ leaves F_{MRP} unchanged.

D.4 Worked example: analytic and minimal-lift chain lifts

For an n-link planar chain with generalized angles $q = (\theta_1, \dots, \theta_n)$ and link lengths ℓ_i, the bob positions are $p_i(q) = \sum_{j=1}^{i} u_j(\theta_j)$ with $u_i(\theta_i) = \ell_i[\sin\theta_i, -\cos\theta_i]^\top$, collected as $X_{\mathrm{chain}}(q) := [p_1(q), \dots, p_n(q)]^\top$. The analytic lift is

$$\phi^{\mathrm{ana}}(q) = \kappa^{\mathrm{ana}}(q) = \mathrm{vec}\big(p_1(q), \dots, p_n(q)\big), \qquad \frac{\partial p_i}{\partial \theta_j} = \mathbf{1}\{j \le i\}\, \ell_j \begin{bmatrix}\cos\theta_j \\ \sin\theta_j\end{bmatrix}. \tag{24}$$

The minimal lift instead exposes one unit rotation vector per joint,

$$s(q) = \mathrm{vec}\big(h(\theta_1), \dots, h(\theta_n)\big), \qquad J_s(q) = \mathrm{blkdiag}\big(Dh(\theta_1), \dots, Dh(\theta_n)\big). \tag{25}$$

The two maps differ only by a fixed equivariant readout: if L_n is the lower-triangular matrix of ones and $Q = \left[\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right]$, then

$$\phi^{\mathrm{ana}}(q) = A_{\mathrm{chain}} s(q), \qquad A_{\mathrm{chain}} = (L_n \otimes \mathrm{I}_2)\, \mathrm{blkdiag}(\ell_1 Q, \dots, \ell_n Q), \tag{26}$$

so $J_{\phi^{\mathrm{ana}}}(q) = A_{\mathrm{chain}} J_s(q)$. Thus the minimal lift retains the equivariant angular chart but omits the known link lengths and cumulative forward-kinematics readout. Both lift variants use $X_{\mathrm{chain}}(q)\hat{g}$ as the input to the gravity branch.

D.5 Lift catalog

Coupled pendulum. For bob i, let $q_i = (\varphi_i, \vartheta_i)$, length ℓ_i, and fixed anchor a_i, and collect the absolute bob positions as

$$X_{\mathrm{coupled}}(q) := \big[a_1 + v_{\ell_1}(q_1), \dots, a_n + v_{\ell_n}(q_n)\big]^\top. \tag{27}$$

The analytic configuration is $\mathrm{vec}(v_{\ell_i}(q_i))_{i=1}^n$, while its absolute kinematic landmarks are $\mathrm{vec}(a_i + v_{\ell_i}(q_i))_{i=1}^n$. The constant anchors do not change the Jacobian. The minimal lift uses the single map

$$s(q) = \mathrm{vec}\big(v_1(q_1), \dots, v_1(q_n)\big), \tag{28}$$

thereby omitting the tether lengths and anchors. Both variants use $X_{\mathrm{coupled}}(q)\hat{g}$ as the input to the gravity branch.

Gyroscope. Let f_1, f_2, f_3 be the rows of $F_{\mathrm{ZXZ}}(q)$, let $\ell_0 = 0$, $\ell_i = f_i$, and let $c_0, \dots, c_3$ be the fixed affine coefficients of the body-frame attachment point. Define

$$p_b = \ell_b - \sum_{j=0}^{3} c_j \ell_j, \qquad X_{\mathrm{gyro}}(q) := \big[p_0, p_1, p_2, p_3\big]^\top, \tag{29}$$

so the analytic map is $\phi^{\mathrm{ana}}(q) = \mathrm{vec}(p_0, p_1, p_2, p_3)$. The minimal lift uses only the frame rows,

$$s(q) = \mathrm{vec}(f_1, f_2, f_3). \tag{30}$$

Both Jacobians follow by differentiating F_{ZXZ}; the analytic map is a fixed linear readout of the minimal map. Both lift variants use $X_{\mathrm{gyro}}(q)\hat{g}$ as the input to the gravity branch.

Rotor and SpacecraftB. For $q = (r, \sigma)$, let f_i be the rows of $F_{\mathrm{MRP}}(\sigma)$ and define

$$\mathcal{C}(\sigma) = \mathrm{vec}(f_1, f_2, f_3), \qquad \mathcal{K}(r, \sigma) = \mathrm{vec}(r, r + f_1, r + f_2, r + f_3). \tag{31}$$

Both minimal lifts use $\phi_c^{\min} = \mathcal{C}$ and $\kappa^{\min} = \mathcal{K}$. SpacecraftB uses the same analytic lift. The Rotor analytic configuration additionally retains an identically zero centered landmark, $\phi_c^{\mathrm{ana}} = \mathrm{vec}(0, f_1, f_2, f_3)$, while using the same $\mathcal{K}$; this changes only a redundant zero feature.

SpacecraftA / Satellite. Let a_x, a_y, a_z be the rows of $F_{\mathrm{MRP}}(\sigma)$, and let $q = (r, \sigma, \theta_1, \theta_2)$. With $c_i = \cos\theta_i$ and $s_i = \sin\theta_i$, define

$$h_1 = r + d_1 a_x, \qquad B_1 = (c_1 a_y + s_1 a_z,\ -s_1 a_y + c_1 a_z,\ a_x), \tag{32}$$

$$h_2 = r - d_2 a_y, \qquad B_2 = (c_2 a_x + s_2 a_z,\ -s_2 a_x + c_2 a_z,\ -a_y). \tag{33}$$

The twelve analytic landmarks are

$$\begin{aligned} P_A(q) = (&r, r + a_x, r + a_y, r + a_z, \\ &h_1, h_1 + B_{1,1}, h_1 + B_{1,2}, h_1 + B_{1,3}, \\ &h_2, h_2 + B_{2,1}, h_2 + B_{2,2}, h_2 + B_{2,3}). \end{aligned} \tag{34}$$

The analytic kinematic map is $\kappa^{\text{ana}} = \text{vec}(P_A)$; its configuration map subtracts the first landmark from all landmarks and drops the resulting zero block. The minimal lift keeps only the bus frame and hinge phases,

$$\phi_c^{\min}(q) = \left[\mathcal{C}(\sigma)^\top, h(\theta_1)^\top, h(\theta_2)^\top\right]^\top, \tag{35}$$

$$\kappa^{\min}(q) = \left[\mathcal{K}(r, \sigma)^\top, h(\theta_1)^\top, h(\theta_2)^\top\right]^\top. \tag{36}$$

The corresponding Jacobians follow by differentiating equations (19) and (23).

Planar N-body system. Writing $q = \text{vec}(x_1, \ldots, x_n) \in \mathbb{R}^{2n}$, the analytic lift is the identity,

$$\phi^{\text{ana}}(q) = \kappa^{\text{ana}}(q) = q, \qquad J_{\text{ana}} = \mathrm{I}_{2n}. \tag{37}$$

The minimal lift removes global position only from the configuration block,

$$\phi_c^{\min}(q) = C_{n,2} q, \qquad \kappa^{\min}(q) = q, \tag{38}$$

with Jacobians $C_{n,2}$ and I_{2n}, respectively.

D.6 Feature dimensions

For completeness, table 6 reports $(\dim V_c, \dim V_k)$, where V_k is the space containing $J_\kappa(q)\dot{q}$. These dimensions count only the inputs to the invariant branch; the auxiliary gravity feature $X(q)\hat{g} \in \mathbb{R}^m$ is not included.

System	$\dim q$	Analytic	Minimal
ChainPendulum(n)	n	$(2n, 2n)$	$(2n, 2n)$
CoupledPendulum(n)	$2n$	$(3n, 3n)$	$(3n, 3n)$
Gyroscope	3	$(12, 12)$	$(9, 9)$
Rotor	6	$(12, 12)$	$(9, 12)$
SpacecraftB	6	$(9, 12)$	$(9, 12)$
SpacecraftA / Satellite	8	$(33, 36)$	$(13, 16)$
NBody(n, planar)	$2n$	$(2n, 2n)$	$(2n, 2n)$
TwoBodySpring	4	$(4, 4)$	$(4, 4)$

Table 6: Configuration- and tangent-feature dimensions of the implemented analytic and minimal lifts.

E Full experimental results

E.1 Results and seed variability

Seed-level uncertainty is small relative to the reported medians. Across all 25 cells of table 1, the standard error of the mean is between 1.6% and 25% of the median, and it is more than an order of magnitude below in 16 of the 25 cells; the exceptions are four Two-body cells (all but the raw LNN), the TwoBodySpring and ChainPendulum EQ-LNN cells, and the Gyroscope EQ-LNN, HNN, and DeLaN cells. Every matched EQ-LNN/lifted-LNN median gap exceeds the combined standard errors of the two cells, so the comparison is seed-robust.

For the minimal-lift control (table 8), the Two-body, Gyroscope, and ChainPendulum cells coincide with the EQ-LNN/lifted-LNN columns of table 1; the Chain minimal-lift EQ-LNN spread (25% of the median, driven by one high seed) is the largest relative spread in the redo.

Table 7: Architecture-family baselines: median trajectory MAE with the standard error of the mean across seeds in parentheses (five seeds for the main-text systems, three for the Rotor and Spacecraft). EQ-LNN uses the same lift as the matched comparison in table 1. CLNN^{P} cells are prior-protocol runs reported without the redo's seed variability (appendix E.3.3); CLNN is a stronger-supervision Cartesian reference (table 9), not a matched causal control. Dashes denote incompatible or unstaged cells rather than imputed results; HNN is omitted on MRP systems because the covariant momentum transformation across shadow switches is not implemented.

System	EQ-LNN	Lifted LNN	Raw LNN	HNN	DeLaN	CLNN^{P}
Two-body	1.77e−2 (2.0e−3)	4.26e−2 (5.9e−3)	3.79e−1 (6.1e−3)	6.86e−2 (8.1e−3)	6.95e−2 (1.3e−2)	1.83e−2
TwoBodySpring	7.90e−5 (1.3e−5)	4.14e−3 (1.1e−4)	6.58e−2 (3.7e−3)	3.51e−2 (1.6e−3)	3.49e−2 (1.2e−3)	1.19e−2
Gyroscope	4.95e−3 (6.2e−4)	1.92e−1 (4.0e−3)	3.01e−1 (1.5e−2)	6.27e−2 (1.2e−2)	6.00e−2 (6.3e−3)	3.24e−4
ChainPendulum	9.02e−4 (2.2e−4)	1.64e−2 (7.4e−4)	4.62e−1 (2.0e−2)	1.02e−1 (4.4e−3)	9.95e−2 (7.2e−3)	1.24e−3
CoupledPendulum	4.68e−2 (1.1e−3)	3.91e−2 (1.0e−3)	1.60e−1 (3.0e−3)	1.51e−1 (2.7e−3)	1.50e−1 (3.4e−3)	3.05e−2
Rotor	1.81e−3 (5.6e−4)	6.73e−2 (5.3e−3)	6.23e−1 (2.0e−2)	–	5.67e−1 (1.5e−2)	3.16e−4
SpacecraftA	6.28e−2 (1.0e−2)	2.04e−1 (2.8e−3)	3.07e−1 (2.8e−3)	–	6.17e−2 (3.5e−3)	1.08e−4
SpacecraftB	1.66e−5 (2.2e−6)	1.33e−3 (1.0e−5)	1.24e−1 (2.3e−3)	–	6.19e−3 (1.9e−3)	1.31e−5

Table 8: Minimal-lift control: median trajectory MAE when the lift, data, optimizer, and seeds are fixed and only the network changes. Main-text cells use five seeds; the Rotor uses three.

System	Minimal-lift EQ-LNN	Minimal-lift LNN
Two-body	1.77×10^{-2}	4.26×10^{-2}
Gyroscope	4.95×10^{-3}	1.92×10^{-1}
ChainPendulum	9.02×10^{-4}	1.64×10^{-2}
Rotor	1.81×10^{-3}	6.73×10^{-2}

The CHNN/CLNN comparison separates supervision from symmetry. CLNN is given redundant Cartesian states, exact constraints, and a narrow mechanical form, whereas EQ-LNN learns a broad invariant scalar Lagrangian from reduced coordinates (table 9). The two families trade off accordingly: EQ-LNN is better on Two-body, ChainPendulum, and TwoBodySpring, while CLNN remains stronger on CoupledPendulum and on the rigid-body systems. We read this as a difference in supplied mechanical information rather than a leaderboard result, and the matched causal conclusions of the paper rest on the lifted-LNN control instead.

E.2 LIFT STUDIES

E.2.1 ANALYTIC AND MINIMAL GEOMETRY

The main-text lift comparison is table 2; its lifted-LNN and raw-LNN cells are the corresponding cells of table 1 and table 8.

Table 9: Full supervision inventory, including the Cartesian-constraint family beyond the reduced-coordinate models of the main text.

Method	Supplied mechanics	Architectural consequence
Raw LNN	Raw coordinates	Variational dynamics; no imposed spatial symmetry
Lifted LNN	Same analytic/minimal lift as EQ-LNN	Isolates feature geometry without exact invariance
EQ-LNN	Group representations and a symmetry-compatible map	Invariant learned Lagrangian and associated Noether laws
CHNN/CLNN	Redundant Cartesian states, exact constraints, their Jacobian, and hard constrained dynamics	Exact constraint satisfaction with stronger mechanistic supervision

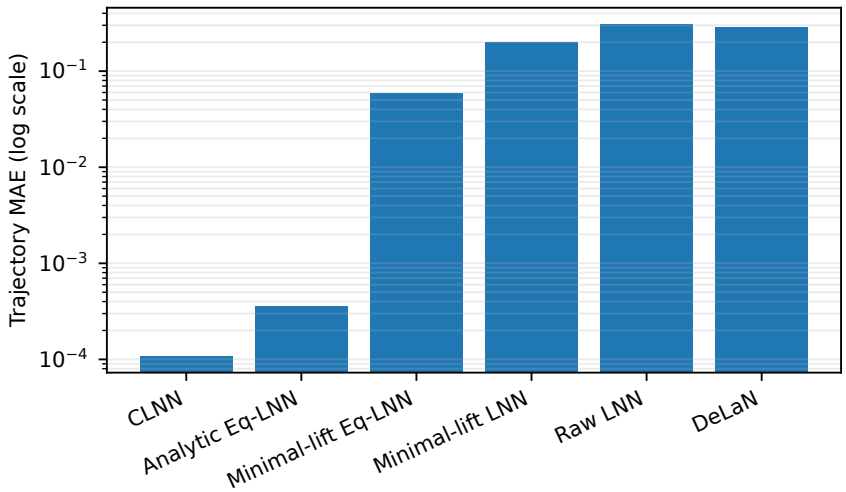

Figure 4: SpacecraftA exposes the limit of minimal geometry. The minimal lift retains equivariance but discards attachment offsets and appendage frames.

E.2.2 LEARNED NOETHER-CHARGE EVALUATION

The evaluation covers 45 checkpoints and 342 run-by-symmetry rows. For each trajectory we compute $J_{\theta,\xi} = \partial_{\dot{q}} L_\theta^\top \xi_{\mathcal{Q}}(q)$ and normalize the maximum drift by $\max(|J(0)|, \epsilon_J)$, where $\epsilon_J = 0.01\,\mathrm{median}|J(0)|$. "Own flow" evaluates the learned trajectory; "floor" evaluates the same learned charge along the true trajectory. Table values are means over three seeds after taking the maximum over the enforced charges in each run. The floor is not a second conservation test because it also depends on momentum-map accuracy and coordinate conditioning; this is most pronounced for the ZXZ Gyroscope, whose L_z floor is 1.73 near gimbal lock, so own-flow drift is the relevant test of theorem 1.

Table 10: Normalized maximum learned-charge drift. The Gyroscope floor is coordinate dominated near ZXZ gimbal lock and is not used as a model-quality comparison.

System	Enforced symmetries	Horizon	EQ-LNN own flow	Lifted LNN	Raw LNN	EQ-LNN floor
Two-body	L_z	20 s	5.07e−6	2.38	7.17	1.69e−2
TwoBodySpring	P_x, P_y, L_z	10 s	1.23e−6	2.37	7.14	2.61e−3
Rotor	$P_{x:y:z}, L_{x:y:z}$	5 s	6.29e−7	3.54	11.0	4.18e−2
Gyroscope	L_z	2 s	2.03e−4	3.85e−1	5.74e−1	1.73*

For TwoBodySpring, the EQ-LNN means are 2.18×10^{-10} for P_x, 1.58×10^{-10} for P_y, and 1.23×10^{-6} for L_z; the corresponding floors are 1.92×10^{-4}, 2.04×10^{-4}, and 2.61×10^{-3}. For

Rotor, the three linear components lie between 4.9×10^{-9} and 1.9×10^{-8} and the three angular components between 1.8×10^{-7} and 6.3×10^{-7}, with floors between 0.007 and 0.042. The lifted LNNs' small linear-momentum drifts on Two-body and the Rotor reflect translation invariance induced by the lifts rather than the equivariant head: both lifted architectures share the lift, so this invariance does not test the scalar network, which is why the conservation comparison centers on the angular components.

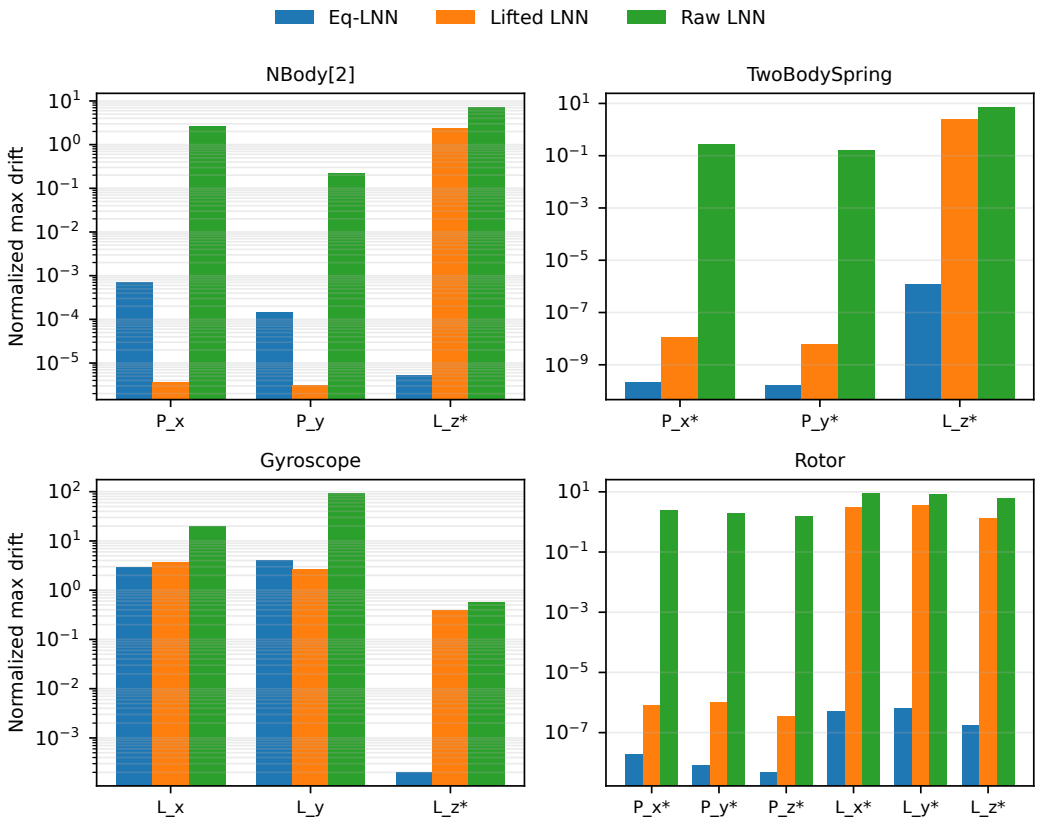

Figure 5: Per-symmetry learned-charge drift for every evaluated system. Bars show the mean normalized maximum drift over seeds for each arm; * marks symmetries enforced by the architecture. Unmarked charges belong to symmetries broken by the construction (for example by fixed gravity) or retained as diagnostics, and are expected to drift. The lifted LNN's small P_x, P_y drift on Two-body reflects translation invariance induced by the centered lift, which both lifted architectures share; it does not test the equivariant scalar head.

Table 11: Gyroscope broken-direction diagnostic for EQ-LNN. Gravity preserves L_z but breaks L_x, L_y.

Generator	Mean normalized max drift	Std. across seeds
L_z (enforced)	2.03e−4	2.56e−4
L_x (broken)	2.84	0.99
L_y (broken)	4.07	1.50

E.2.3 FREE ROTATIONS WITH MRP SHADOW COORDINATES

The original ZXZ realization becomes poorly conditioned near gimbal lock, so the free spatial bodies use modified Rodrigues parameters (MRPs). For a nonzero MRP vector σ, the equivalent shadow

coordinate and its Jacobian are

$$S(\sigma) = -\frac{\sigma}{\|\sigma\|^2}, \qquad DS(\sigma) = -\frac{I}{\|\sigma\|^2} + \frac{2\sigma\sigma^\top}{\|\sigma\|^4}.$$

The implementation verifies $R(S(\sigma)) = R(\sigma)$ and transforms rates by $\dot{\sigma}^S = DS(\sigma)\dot{\sigma}$. Consequently, the physical lifted velocity obeys $J_\kappa(S(\sigma))DS(\sigma)\dot{\sigma} = J_\kappa(\sigma)\dot{\sigma}$. Rollouts integrate one output interval at a time and canonicalize between intervals, while losses are evaluated in the physical embedding; the computational graph is preserved within each integration segment. This is not a claim that MRPs form one global coordinate system; rather, the shadow pair gives a bounded switched-coordinate realization whose lifted features are invariant to the switch.

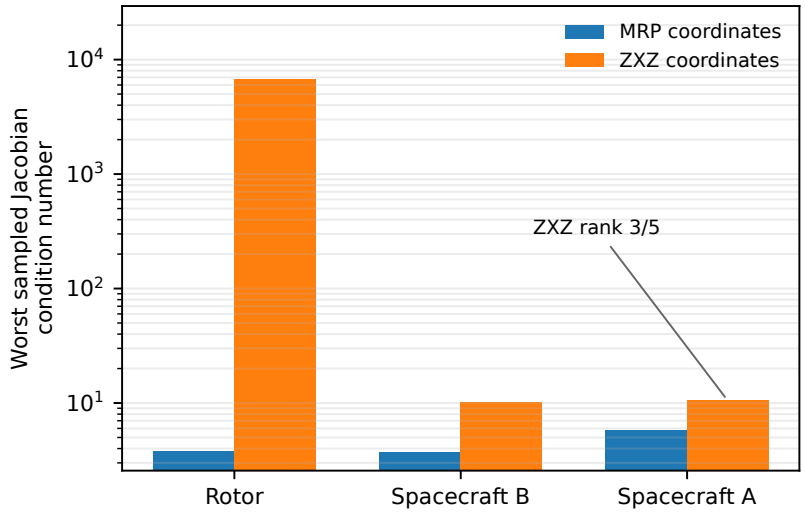

Figure 6: Worst sampled Jacobian condition number over 4,096 states. SpacecraftA's sampled ZXZ map also has rank 3 versus expected rank 5, while MRP has rank 5.

Table 12: MRP preflight. All bodies pass the minimum-singular-value gate 10^{-3} and reach the shadow-switch region during true rollouts.

System	MRP $\sigma_{\min}^{\min}$	MRP $\kappa_{\max}$	ZXZ $\sigma_{\min}^{\min}$	ZXZ $\kappa_{\max}$
Rotor	1.414	3.76	3.65e−4	6.73e3
SpacecraftB	1.414	3.73	0.241	10.21
SpacecraftA	1.000	5.77	0.235	10.46

E.3 Diagnostics and prior-protocol artifacts

E.3.1 Physical invariant fidelity

The trainer's staged drift metrics evaluate the true-body Hamiltonian and true-body linear/angular momentum along predicted trajectories. They answer whether the learned trajectory respects the physical invariants, not whether it conserves its own $J_{\theta,\xi}$. As one representative case, TwoBodySpring median integrated log drifts (E, L, P) are $(-112.1, -99.4, -105.5)$ for Eq-LNN, $(-80.9, -63.8, -72.0)$ for the matched lifted LNN, and $(-57.7, -42.9, -50.5)$ for raw LNN; lower is better.

E.3.2 Symmetry OOD generalization

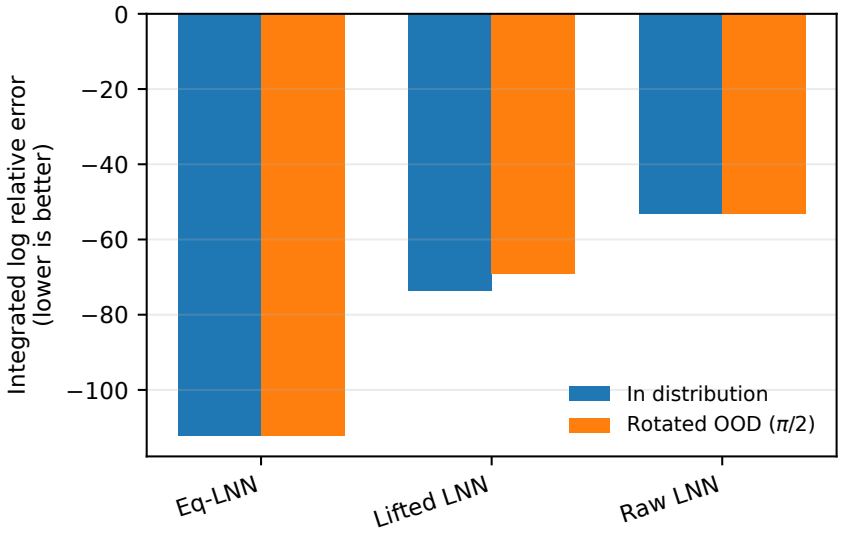

Figure 7: TwoBodySpring in-distribution and $\pi/2$ rotation-OOD error. Training orientations lie in a narrow ± 0.25 rad sector. Exact equivariance makes the EQ-LNN result unchanged to the reported digits, while both non-equivariant models degrade.

E.3.3 Prior-protocol artifacts

All matched-training cells now come from the final protocol. Three result families still rest on the earlier training runs and are marked where used: the CLNN column of table 7 (CLNN was not rerun), the learned-charge evaluation of table 10 (45 checkpoints, three seeds each), and the morphology ablation below.

E.3.4 Morphology ablation

This ablation predates the final protocol and uses three seeds (appendix E.3.3); the ChainPendulum minimal-lift instability discussed below is specific to that protocol's learning rate and is no longer present under the final one (table 8). Comparing analytic and minimal-lift EQ-LNN runs, ChainPendulum gives $8.4\text{e}{-4}$ versus $8.7\text{e}{-4}$ over the finite minimal-lift seeds, with one of three minimal-lift seeds diverging at 1.05 under the frozen protocol; the unequal-length CoupledPendulum variant gives $2.78\text{e}{-2}$ versus $2.65\text{e}{-2}$, and the Gyroscope gives $7.04\text{e}{-3}$ versus $4.29\text{e}{-3}$. The final same-lift controls show that the Chain instability is not shared by the lifted LNN at this protocol, whereas the large Gyroscope accuracy gain remains attributable to the equivariant network.

F Method code listings

This appendix reproduces the method-defining code as trimmed, argument-ordered Python listings. Every listing compiles standalone (`python -m py_compile` verified); symbols either resolve in a prior listing or are shown with their real-repository import path as provenance. Docstrings state the construction invariants.

Provenance. The listings are the current working tree (2026-09-21) — the same code that produced the paper's results. Excerpted listings state their source file in the header comment. The code names the minimal lift *skeleton*; listing titles preserve those identifiers.

What is not listed. The intertwiner basis solve and the generic equivariant layers are the standard arbitrary-matrix-group construction of Finzi et al. (2021); their mathematical specification is given in section 3.2, appendix B.2, and appendix B, so the corresponding listings are omitted. Also omitted:

training drivers and sweep plumbing, baseline models (standard HNN/LNN/DeLaN/CHNN/CLNN), the per-body analytic coordinate modules (their mathematical definitions are given in appendices D.3 and D.5), and generic validation raises elided with markers at their sites. The data-generation, evaluation-protocol, and per-system instantiation tables that accompany these listings appear in appendices B and C.

Table 13: Code listings and their role in the argument.

Listing	Source	Role in the argument
1	Lift specifications	The `LiftSpec`/`FieldSpec` vocabulary every later listing consumes.
2	Gravity projection	`GravityProjection` (the $x(q) \cdot g$ module) and its wiring into every analytic lift via `make_cartesian_lift`.
3	Equivariant skeletons	The minimal equivariant maps $s(q)$ underlying the minimal lift.
4	Skeleton (split) lift	The headline construction: the minimal split lift — invariant configuration map $s_c(q)$ for the energy, absolute landmark kinematics $J_{\text{sk}}(q)\,\dot{q}$ for the kinetic energy.
5	Noether charge	The evaluation instrument behind the conservation claims: symmetry generators as data, learned momentum, charge along the model's own rollout, and robust drift normalization.

Reading order follows the paper's argument chain: the lift interface (listing 1) → the gravity branch (listing 2) → the minimal-lift construction (listings 3 and 4) → the conservation evaluation (listing 5).

Listing 1: Lift specifications

```python
# ============================================================================
# APPENDIX LISTING 3 --- LIFT SPEC
# ============================================================================
# The vocabulary: FieldSpec / RepresentationSpec / LiftSpec --- the dataclasses carrying a lift's
    configuration map, kinematics, group representation, and gravity module. Every later listing consumes
    these.
#
# Source: source/models/equivariant_parts/specs.py.
# Working tree 2026-09-21 --- the same code that produced the paper results.
# ============================================================================

from dataclasses import dataclass
from typing import Callable, Optional, Tuple, Union

import torch
import torch.nn as nn
from torch import Tensor


def as_double_tensor(value) -> Tensor:
    return torch.as_tensor(value, dtype=torch.float64).detach().cpu()


@dataclass(frozen=True)
class FieldSpec:
    indices: Tuple[int, ...]
    metric: Optional[Tensor] = None

    def __post_init__(self):
        # nonempty-indices validation elided
        object.__setattr__(self, "indices", tuple(self.indices))
        if self.metric is not None:
            object.__setattr__(self, "metric", as_double_tensor(self.metric))


@dataclass(frozen=True)
class RepresentationSpec:
    generators: Tensor
    scalar_indices: Tuple[int, ...] = tuple()
    fields: Tuple[FieldSpec, ...] = tuple()
    name: str = "representation"
    tolerance: float = 1e-7

    def __post_init__(self):
        # shape/tolerance and partition/block-invariance validation elided
        object.__setattr__(self, "generators", as_double_tensor(self.generators))
        object.__setattr__(self, "scalar_indices", tuple(self.scalar_indices))
        object.__setattr__(self, "fields", tuple(self.fields))

    @property
    def dimension(self) -> int:
        return self.generators.shape[-1]
```

```
    @property
    def num_generators(self) -> int:
        return self.generators.shape[0]


@dataclass(frozen=True)
class RepresentationLayout:
    representation: RepresentationSpec
    multiplicity: int = 1
    name: Optional[str] = None

    @property
    def dimension(self) -> int:
        return self.representation.dimension * self.multiplicity

    @property
    def representation_dimension(self) -> int:
        return self.representation.dimension

    @property
    def num_generators(self) -> int:
        return self.representation.num_generators


RepresentationLike = Union[RepresentationSpec, RepresentationLayout]


def as_representation_layout(value: RepresentationLike) -> RepresentationLayout:
    if isinstance(value, RepresentationLayout):
        return value
    if isinstance(value, RepresentationSpec):
        return RepresentationLayout(value)
    raise TypeError("expected a RepresentationSpec or RepresentationLayout")


class _FunctionModule(nn.Module):
    def __init__(self, function: Callable[[Tensor], Tensor]):
        super().__init__()
        self.function = function

    def forward(self, inputs: Tensor) -> Tensor:
        return self.function(inputs)


def _as_module(value):
    return value if isinstance(value, nn.Module) else _FunctionModule(value)


@dataclass(frozen=True)
class LiftSpec:
    configuration: nn.Module
    representation: RepresentationSpec
    generalized_dimension: Optional[int] = None
    kinematics: Optional[nn.Module] = None
    tangent_representation: Optional[RepresentationSpec] = None
    jacobian: Optional[Callable[[Tensor], Tensor]] = None
    gravity_features: Optional[nn.Module] = None

    def __post_init__(self):
        configuration = _as_module(self.configuration)
        kinematics = configuration if self.kinematics is None else _as_module(self.kinematics)
        tangent_representation = (
            self.representation
            if self.tangent_representation is None
            else self.tangent_representation
        )
        gravity_features = (
            None
            if self.gravity_features is None
            else _as_module(self.gravity_features)
        )
        jacobian = self.jacobian
        if jacobian is None and hasattr(kinematics, "jacobian"):
            jacobian = kinematics.jacobian
        # consistency validation elided (matching generators, positive
        # generalized_dimension)
        object.__setattr__(self, "configuration", configuration)
        object.__setattr__(self, "kinematics", kinematics)
        object.__setattr__(self, "tangent_representation", tangent_representation)
        object.__setattr__(self, "jacobian", jacobian)
        object.__setattr__(self, "gravity_features", gravity_features)
```

Listing 2: Gravity projection

```
# ============================================================================
# APPENDIX LISTING 5 --- GRAVITY PROJECTION
# ============================================================================
# GravityProjection: the external-field module x(q) . g consumed by the nets' gravity branch (listing 4), and
      its wiring into every analytic lift via make_cartesian_lift. The per-body gravity_positions maps live in
       coordinates.py / systems.py (not reproduced here).
```

```
#
# Source: source/equivariant_lifts/core.py.
# Working tree 2026-09-21 --- the same code that produced the paper results.
# ============================================================================

class GravityProjection(nn.Module):
    def __init__(self, positions: nn.Module, vector_count: int, spatial_dimension: int):
        super().__init__()
        self.positions = positions
        self.vector_count = vector_count
        self.spatial_dimension = spatial_dimension
        self.output_dimension = vector_count
        gravity = torch.zeros(spatial_dimension, dtype=torch.float64)
        gravity[-1] = -1.0
        self.register_buffer("gravity", gravity)

    def forward(self, q: Tensor) -> Tensor:
        positions = self.positions(q).reshape(
            q.shape[0], self.vector_count, self.spatial_dimension
        )
        return positions @ self.gravity.to(q)


@export


# ============================================================================

# -- wiring into every analytic lift (make_cartesian_lift) --

def make_cartesian_lift(configuration, kinematics, gravity_positions,
                        tangent_vector_count, generators):
    """Gravity-feature wiring excerpt --- the rest of make_cartesian_lift
    (metadata validation, representation construction) is omitted here.
    """
    gravity_features = None
    if gravity_positions is not None:
        gravity_vector_count = getattr(
            gravity_positions, "vector_count", tangent_vector_count
        )
        # count validation elided: one gravity position per tangent vector
        gravity_features = GravityProjection(
            gravity_positions,
            tangent_vector_count,
            generators.shape[-1],
        )
```

Listing 3: Equivariant skeletons

```
# ============================================================================
# APPENDIX LISTING 6 --- SKELETONS
# ============================================================================
# The minimal skeletons: PlanarAngleSkeleton / ZXZFrameSkeleton --- minimal equivariant feature maps s(q)
     underlying the skeleton lift (listing 7).
#
# Source: source/equivariant_lifts/skeletons.py.
# Working tree 2026-09-21 --- the same code that produced the paper results.
# ============================================================================

"""Skeleton feature maps for the equivariant lifts.

Each skeleton is a *minimal, exactly equivariant* feature map ``s(q)`` that
exposes a body's chart structure without hand-derived Cartesian geometry.
Every module declares ``vector_count``/``generalized_dimension`` metadata
(validated by ``make_cartesian_lift``) and provides an analytic ``jacobian``
returning ``[batch, vector_count * spatial_dim, dof]``; forward and jacobian
assemble out-of-place so they stay traceable under ``vmap``.
"""

import torch
import torch.nn as nn
from torch import Tensor

from source.equivariant_lifts.coordinates import _zxz_frame_and_derivatives
from source.utils import export


@export
class PlanarAngleSkeleton(nn.Module):
    """Unchained per-angle ``(cos theta_i, sin theta_i)`` features.

    The chain-pendulum skeleton: one rotation vector per angle --- lengths and
    chaining live in the fixed intertwiner ``A`` with
    ``phi_analytic = A s(q)``, so this skeleton's feature class contains the
    analytic chain lift.  Under the SO(2) chart translation each 2-vector
    rotates: ``repeated_spatial_representation(spatial_rotation_generators(2), n)``.
    """

    def __init__(self, num_angles: int):
        super().__init__()
```

```
        self.vector_count = int(num_angles)
        self.generalized_dimension = self.vector_count

    def forward(self, q: Tensor) -> Tensor:
        return torch.stack([q.cos(), q.sin()], dim=-1).reshape(q.shape[0], -1)

    def jacobian(self, q: Tensor) -> Tensor:
        batch_size, dof = q.shape[0], q.shape[-1]
        # d(cos theta_i)/d theta_j = -sin(theta_i) delta_ij,
        # d(sin theta_i)/d theta_j =  cos(theta_i) delta_ij: one nonzero
        # 2x1 block per column, assembled out-of-place (vmap-safe).
        columns = []
        for index in range(dof):
            block = torch.stack(
                [-q[:, index].sin(), q[:, index].cos()], dim=-1
            ).unsqueeze(-1)
            left = torch.zeros(
                batch_size, 2 * index, 1, device=q.device, dtype=q.dtype
            )
            right = torch.zeros(
                batch_size, 2 * (dof - index - 1), 1,
                device=q.device, dtype=q.dtype,
            )
            columns.append(torch.cat([left, block, right], dim=-2))
        return torch.cat(columns, dim=-1)


@export
class ZXZFrameSkeleton(nn.Module):
    """Raw ZXZ body-frame rows: the gyroscope's minimal SO(3) skeleton.

    ``forward`` flattens the three body-axis world-coordinate vectors from
    ``_zxz_frame_and_derivatives`` into a 9-dimensional feature.  Under a
    spatial rotation the rows transform as ``F' = F g^T`` (each rotates
    independently): ``repeated_spatial_representation(spatial_rotation_generators(3), 3)``.
    Unlike the ZXZ chart itself, the frame rows are globally valid --- the
    beta = 0/pi poles remain in the *chart* q, where only Jacobian rank can drop.
    """

    vector_count = 3
    generalized_dimension = 3

    def forward(self, q: Tensor) -> Tensor:
        frame, _ = _zxz_frame_and_derivatives(q)
        return frame.reshape(q.shape[0], -1)

    def jacobian(self, q: Tensor) -> Tensor:
        _, derivatives = _zxz_frame_and_derivatives(q)
        # derivatives[b, k, i, j] = d frame[b, i, j] / d euler_k, so the
        # flattened-feature Jacobian is J[b, 3*i + j, k].
        return derivatives.permute(0, 2, 3, 1).reshape(q.shape[0], 9, 3)
```

Listing 4: Skeleton (split) lift

```
# ============================================================================
# APPENDIX LISTING 7 --- SKELETON LIFT
# ============================================================================
# The headline construction: skeleton_lift's split LiftSpec --- an invariant configuration map s_c(q) feeding
      the energy, with absolute landmark kinematics J_sk(q) qdot carrying the kinetic energy, so the lifted
      LNN consumes exactly [s_c(q), J_sk(q) qdot].
#
# Source: source/equivariant_lifts/skeleton_lifts.py.
# Working tree 2026-09-21 --- the same code that produced the paper results.
# ============================================================================

"""Skeleton lifts for the free bodies (no learned-map arms).

A skeleton lift is the *split-LiftSpec* construction for bodies whose
configuration and kinematics feature classes differ: the configuration map is
the minimal invariant skeleton ``s_c(q)`` (translation-invariant) while the
kinematics keep the absolute landmark map ``J_sk(q) qdot`` that carries the
kinetic energy.  The lifted LNN consumes exactly ``[s_c(q), J_sk(q) qdot]``
(``LiftedLagrangianNet.forward``), giving the architecture

    L_theta(q, qdot) = ell_theta(s_c(q), J_sk(q) qdot)

These are fixed maps: no learned feature-map parameters, LNN-only (the
configuration/kinematics split presumes the Lagrangian input layout), and
zero-potential systems, so the ``gravity`` flag is accepted for signature
uniformity and ignored.
"""

from source.equivariant_lifts.coordinates import (
    FreePointMassCoordinates,
    MRPFrameCoordinates,
    MRPFramePhaseCoordinates,
)
from source.equivariant_lifts.core import (
    make_free_cartesian_lift,
    spatial_rotation_generators,
```

```
)
from source.models.equivariant import (
    LiftSpec,
    direct_sum_representation,
    repeat_representation,
    trivial_representation,
)
from source.models.equivariant import LiftSpec
from source.utils import export

# The free bodies with registered skeleton entries.  Kept distinct from
# LEARNED_LIFT_BODIES (the anchored learned-map bodies) on purpose: the two
# allowlists gate different modes and must not grow in lockstep.
SKELETON_LIFT_BODIES = frozenset(
    {"Rotor", "NBody", "TwoBodySpring", "SpacecraftA", "SpacecraftB", "Satellite"}
)


@export
def skeleton_lift(body, gravity: bool = True) -> LiftSpec:
    """Skeleton LiftSpec for a free body (``chart_invariant_errors`` bodies + point masses).

    ``gravity`` is accepted for signature uniformity with the analytic lift
    factories but is inapplicable here --- every skeleton body is zero-potential
    or carries an internal-only potential, so there is no external field to
    project.  It is intentionally ignored.
    """
    body_name = type(body).__name__
    if body_name in ("Rotor", "SpacecraftB"):
        # q = [com(3), sigma(3)]; configuration = the three body-frame axis
        # rows (translation-invariant), kinematics = center + axis-endpoint
        # landmarks (absolute, so the kinetic energy survives centering).
        configuration = MRPFrameCoordinates(relative=True, include_center=False)
        kinematics = MRPFrameCoordinates(relative=False, include_center=True)
        return make_free_cartesian_lift(
            configuration,
            kinematics,
            spatial_rotation_generators(3),
            name=f"{body_name.lower()}_skeleton",
        )
    if body_name in ("SpacecraftA", "Satellite"):
        # q = [r(3), sigma(3), theta_1, theta_2].  The configuration keeps the
        # three bus-axis rows (translation-invariant) plus the four hinge phase
        # scalars; the kinematics keep every absolute bus landmark plus the
        # phases, so the kinetic energy is represented exactly.  The mixed
        # representation --- repeated SO(3) vectors on the frame block, trivial
        # scalars on the phase block --- needs a direct LiftSpec (the
        # make_*_cartesian_lift helpers only build repeated-vector reps).
        configuration = MRPFramePhaseCoordinates(relative=True, include_center=False)
        kinematics = MRPFramePhaseCoordinates(relative=False, include_center=True)
        if body.n != kinematics.full_vector_count * 3 or body.D != 8:
            raise ValueError(
                f"{body_name} has n={body.n}, D={body.D}; the skeleton entry "
                f"expects n=12, D=8"
            )
        generators = spatial_rotation_generators(3)
        representation = direct_sum_representation(
            repeat_representation(generators, 3),
            trivial_representation(generators.shape[0], 4),
        )
        tangent_representation = direct_sum_representation(
            repeat_representation(generators, 4),
            trivial_representation(generators.shape[0], 4),
        )
        return LiftSpec(
            configuration=configuration,
            representation=representation,
            generalized_dimension=body.D,
            kinematics=kinematics,
            tangent_representation=tangent_representation,
            jacobian=kinematics.jacobian,
        )
    if body_name in ("NBody", "TwoBodySpring"):
        # q = flattened Cartesian point positions (body-major, d-inner ---
        # NBody's generalized coordinates ARE Cartesian; TwoBodySpring's
        # layout matches).  Configuration is the centered point map
        # (translation-invariant; for NBody strictly more invariant than its
        # analytic identity lift), kinematics keeps absolute positions with
        # the identity Jacobian so COM kinetic energy survives.
        if body_name == "NBody" and body.d != 2:
            raise ValueError(
                f"NBody skeleton entry expects the planar system (d=2), got d={body.d}"
            )
        configuration = FreePointMassCoordinates(body.n, body.d, relative=True)
        kinematics = FreePointMassCoordinates(body.n, body.d, relative=False)
        return make_free_cartesian_lift(
            configuration,
            kinematics,
            spatial_rotation_generators(2),
            generalized_dimension=body.D,
            name=f"{body_name.lower()}_skeleton",
        )
```

```
    raise ValueError(
        f"skeleton_lift has no entry for body class {body_name!r}; supported "
        f"bodies are {sorted(SKELETON_LIFT_BODIES)}"
    )
```

Listing 5: Learned Noether-charge evaluation

```
# ============================================================================
# APPENDIX LISTING 9 --- NOETHER CHARGE
# ============================================================================
# The evaluation instrument behind the conservation claims: symmetry generators, the learned momentum p_theta
    = dL/dqdot, the learned charge J = p . xi_Q(q) along the model's own rollout, drift metrics, and robust
    normalization.
#
# Source: source/noether.py.
# Working tree 2026-09-21 --- the same code that produced the paper results.
# ============================================================================

"""Learned Noether-charge evaluation for generalized-coordinate energy models.

For a learned Lagrangian ``L_theta(q, qdot)`` the momentum map of a symmetry
with infinitesimal generator ``xi`` is

    J_{theta, xi}(q, qdot) = p_theta(q, qdot)^T xi_Q(q),
    p_theta = d L_theta / d qdot,

and for a system whose action on the Cartesian embedding ``x = f(q)`` of the
node positions is ``delta x = A x + a`` (``A`` antisymmetric for rotations,
``A = 0`` for translations), the induced generator field on generalized
coordinates solves the pushforward equation

    J_f(q) xi_Q(q) = A x(q) + a,      J_f(q) = d x / d q.

``J_f`` is obtained WITHOUT autodiff through ``body2globalCoords`` (which is
scipy-based for the euler chart): by the kinematic convention of the
``(bs, 2, ...)`` state layout, the velocity channel of ``body2globalCoords``
is exactly ``J_f(q) qdot``, so unit rate vectors give the Jacobian columns.

For LNN the momentum is ``autograd`` of the model's own scalar ``L`` (the same
function its ``LagrangianDynamics`` integrates, including the ``eps||J qdot||^2``
regularizer, so the charge is the one conserved by the learned flow); for the
``GeneralizedEnergyNet`` family (HNN, DirectDeLaN) ``p_theta = M(q) qdot``.

The charge is a scalar pairing (covector ``p`` against a vector field), so it
is invariant under chart changes --- MRP shadow switches during rollouts do not
affect its value.

Generator registries are declared per system as DATA (linear actions on node
positions), never as system-specific charge formulas.  ``conserved=False``
marks generators whose symmetry is broken (e.g. rotations about the horizontal
axes of the Gyroscope, whose gravity picks out z): they are computed for
diagnostic figures of the broken directions and must not be presented as
conserved charges.
"""
from dataclasses import dataclass

import torch

from source.models.hnn import HNN
from source.models.lnn import DirectDeLaN, LNN


@dataclass(frozen=True)
class SymmetryGenerator:
    """Infinitesimal action ``delta x_n = A x_n + a`` on node positions."""

    name: str
    A: torch.Tensor  # (d, d) infinitesimal linear action
    a: torch.Tensor  # (d,) infinitesimal translation part
    conserved: bool = True  # exact symmetry of the physical system?
    note: str = ""


def cross_matrix(axis: torch.Tensor) -> torch.Tensor:
    """The so(d) generator ``A`` with ``A x = axis x x`` (d = 2 or 3)."""
    if axis.shape == ():
        raise ValueError("axis must be a d-vector")
    d = axis.numel()
    axis = axis / axis.norm()
    if d == 2:
        # rotation about z: delta x = (-y, x)  ->  L_z = x p_y - y p_x
        return torch.tensor([[0.0, -1.0], [1.0, 0.0]], dtype=axis.dtype)
    if d == 3:
        ax, ay, az = axis.tolist()
        return torch.tensor(
            [[0.0, -az, ay], [az, 0.0, -ax], [-ay, ax, 0.0]], dtype=axis.dtype
        )
    raise NotImplementedError(f"cross_matrix undefined for d={d}")
```

```python
def rotation_generator(name: str, axis, d: int, conserved: bool = True, note: str = ""):
    """Rotation about ``axis`` (through the origin) in ``d`` dimensions.

    ``d == 2`` systems use the planar generator ``[[0, -1], [1, 0]]`` (rotation
    about z) regardless of the ``axis`` argument's spelling.
    """
    if d == 2:
        A = cross_matrix(torch.tensor([0.0, 1.0], dtype=torch.float64))
    else:
        A = cross_matrix(torch.as_tensor(axis, dtype=torch.float64))
    return SymmetryGenerator(
        name=name, A=A, a=torch.zeros(d), conserved=conserved, note=note
    )


def translation_generator(name: str, axis: int, d: int, conserved: bool = True, note: str = ""):
    """Unit translation along ``axis`` (0-indexed)."""
    a = torch.zeros(d, dtype=torch.float64)
    a[axis] = 1.0
    return SymmetryGenerator(
        name=name, A=torch.zeros(d, d, dtype=torch.float64), a=a,
        conserved=conserved, note=note
    )


#: Per-system generator inventories, keyed by body class name.  ``conserved``
#: follows the pinned invariants table in ``tests/test_conservation.py``:
#: gravity along z keeps only the stabilizer rotation L_z for the Gyroscope;
#: the anchored pendula have NO continuous symmetry beyond energy (so they are
#: absent); free bodies (Rotor, TwoBodySpring, NBody) conserve P and L.
SYSTEM_GENERATORS = {
    "Gyroscope": lambda d: [
        rotation_generator("L_z", [0.0, 0.0, 1.0], d, conserved=True,
                           note="stabilizer of gravity axis (surviving symmetry)"),
        rotation_generator("L_x", [1.0, 0.0, 0.0], d, conserved=False,
                           note="broken by gravity --- diagnostic only"),
        rotation_generator("L_y", [0.0, 1.0, 0.0], d, conserved=False,
                           note="broken by gravity --- diagnostic only"),
    ],
    "TwoBodySpring": lambda d: [
        translation_generator("P_x", 0, d, note="internal spring, no anchors"),
        translation_generator("P_y", 1, d, note="internal spring, no anchors"),
        rotation_generator("L_z", [0.0, 0.0, 1.0], d, note="isotropic spring in the plane"),
    ],
    "Rotor": lambda d: [
        translation_generator("P_x", 0, d),
        translation_generator("P_y", 1, d),
        translation_generator("P_z", 2, d),
        rotation_generator("L_x", [1.0, 0.0, 0.0], d),
        rotation_generator("L_y", [0.0, 1.0, 0.0], d),
        rotation_generator("L_z", [0.0, 0.0, 1.0], d),
    ],
    "NBody": lambda d: [
        # All bodies share one planar rotation about the origin (COM sits at
        # the origin in the data); translations are physically conserved but
        # the rotation-only lift does not represent them (see the runner's
        # architectural column) --- compute them as diagnostics.
        translation_generator("P_x", 0, d, note="conserved physically; check lift representation"),
        translation_generator("P_y", 1, d, note="conserved physically; check lift representation"),
        rotation_generator("L_z", [0.0, 0.0, 1.0], d, note="planar rotation about the COM"),
    ],
}


def system_generators(body):
    """The generator inventory declared for ``body``'s class."""
    key = type(body).__name__
    if key not in SYSTEM_GENERATORS:
        raise KeyError(
            f"no generator inventory registered for {key!r}; add it to "
            f"source.noether.SYSTEM_GENERATORS (generators are declared as "
            f"data, not formulas)"
        )
    return SYSTEM_GENERATORS[key](body.d)


def embedding_positions(body, q):
    """Cartesian node positions ``x(q)``: (bs, D) -> (bs, n, d).

    ``q`` is the model-space generalized position batch; the rates channel is
    zero (positions do not depend on rates).
    """
    D = body.D
    q = q.reshape(-1, D)
    rates = torch.zeros_like(q)
    state = torch.stack([q, rates], dim=1)  # (bs, 2, D)
    out = body.body2globalCoords(state)
    return out[:, 0]


def embedding_jacobian(body, q):
    """``J_f(q) = d x / d q`` as (bs, n*d, D), from the kinematic rate channel.
```

```
    The velocity channel of ``body2globalCoords`` is ``J_f(q) qdot`` by the
    ``(pos, rate)`` state convention, so unit rate vectors give the exact
    Jacobian columns with pure forward evaluation (no autodiff through the
    scipy-based euler chart).
    """
    D = body.D
    q = q.reshape(-1, D)
    columns = []
    for k in range(D):
        rates = torch.zeros_like(q)
        rates[:, k] = 1.0
        state = torch.stack([q, rates], dim=1)  # (bs, 2, D)
        out = body.body2globalCoords(state)
        columns.append(out[:, 1].reshape(q.shape[0], -1))
    return torch.stack(columns, dim=-1)  # (bs, n*d, D)


def generator_field(body, generator: SymmetryGenerator, q, jacobian=None):
    """``xi_Q(q)`` solving ``J_f(q) xi_Q = A x(q) + a`` (bs, D).

    The system is over-determined on the ambient node space but consistent:
    the rigid-body constraint manifold's tangent space at ``x(q)`` is spanned
    by the embedding Jacobian's columns, and ``A x + a`` lies in it for every
    group action (rotations/translations preserve the constraints).
    """
    x = embedding_positions(body, q)  # (bs, n, d)
    bs = x.shape[0]
    action = (x @ generator.A.transpose(0, 1) + generator.a).reshape(bs, -1)
    if jacobian is None:
        jacobian = embedding_jacobian(body, q)
    xi = torch.linalg.lstsq(jacobian, action.unsqueeze(-1), driver="gels").solution
    return xi.squeeze(-1)


def learned_momentum(model, q, qdot):
    """``p_theta = dL/dqdot`` (bs, D) for the supported learned scalar objects.

    LNN: autodiff through the model's own scalar ``L`` --- the same function its
    dynamics integrate (including the kinetic regularizer), so the resulting
    charge is the one the learned flow conserves.  HNN / DirectDeLaN: the
    learned (generalized) mass applied to the velocity, ``p = M(q) qdot``.
    """
    if isinstance(model, LNN):
        z = torch.cat([q, qdot], dim=-1)
        z = z.detach().requires_grad_(True)
        lagrangian = model.L(torch.zeros((), dtype=z.dtype, device=z.device), z)
        (grad,) = torch.autograd.grad(lagrangian.sum(), z)
        D = q.shape[-1]
        return grad[:, D:]
    if isinstance(model, (HNN, DirectDeLaN)):
        if isinstance(model, DirectDeLaN):
            mass = model.mass_matrix(q)
            return (mass @ qdot.unsqueeze(-1)).squeeze(-1)
        inverse_mass = model.Minv(q)
        return torch.linalg.solve(inverse_mass, qdot.unsqueeze(-1)).squeeze(-1)
    raise TypeError(
        f"no learned-momentum accessor for {type(model).__name__}; supported: "
        f"LNN (autodiff dL/dqdot), HNN and DirectDeLaN (M(q) qdot)"
    )


def learned_charge(model, body, generator: SymmetryGenerator, q, qdot, xi=None):
    """``J_{theta, xi} = p_theta . xi_Q(q)`` per trajectory (bs,)."""
    if xi is None:
        xi = generator_field(body, generator, q)
    p = learned_momentum(model, q, qdot)
    return (p * xi).sum(-1)


def charges_along_trajectory(model, body, generators, zts):
    """Learned charges at every step of a model-space rollout.

    ``zts``: (T, bs, 2D) in the model's generalized coordinates.  Returns
    ``{generator.name: (T, bs) float64 charges}``.
    """
    T = zts.shape[0]
    qts, vts = zts.chunk(2, dim=-1)
    out = {generator.name: [] for generator in generators}
    for t in range(T):
        q, v = qts[t].double(), vts[t].double()
        jacobian = embedding_jacobian(body, q)
        p = learned_momentum(model, q, v)
        for generator in generators:
            xi = generator_field(body, generator, q, jacobian=jacobian)
            out[generator.name].append((p * xi).sum(-1))
    return {name: torch.stack(values, dim=0) for name, values in out.items()}


def drift_metrics(charge: torch.Tensor):
    """Per-trajectory drift statistics of ``J(t) - J(0)``.
```

```
    ``charge``: (T, bs).  Returns dict of (bs,) tensors: max/rms/mean/final of
    the absolute drift.
    """
    delta = charge - charge[:1]
    return {
        "max_abs_drift": delta.abs().amax(dim=0),
        "rms_drift": delta.pow(2).mean(dim=0).sqrt(),
        "mean_abs_drift": delta.abs().mean(dim=0),
        "final_abs_drift": delta[-1].abs(),
    }


def robust_normalize(drift: torch.Tensor, j0: torch.Tensor, eps_frac: float = 1e-2):
    """Drift normalized by ``max(|J(0)|, eps_J)`` per trajectory.

    ``eps_J = eps_frac * median(|J(0)|)`` over the trajectory set guards
    against the near-zero-charge blowup of a naive ``|J(0)|`` denominator.
    ``torch.quantile`` (not ``torch.median``, which returns the lower element
    for even-length inputs) so a half-zero charge set still gets a floor.
    """
    eps_j = eps_frac * torch.quantile(j0.abs(), 0.5)
    denominator = j0.abs().clamp(min=eps_j)
    return drift / denominator
```