JUMPLIE: DISCOVERING AND EXPLOITING JUMP-DIFFUSION SYMMETRIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Jump-diffusion (JD) processes provide a natural description of stochastic systems that combine continuous evolution with abrupt discrete events, a setting increasingly relevant to machine learning, while symmetry offers a powerful means of exposing structure and constructing inductive biases for learning. We introduce *JumpLie*, an end-to-end, data-driven framework for discovering and exploiting Lie symmetries of JD dynamics, building on the point-symmetry theory of A. Nass (2017). Our approach first learns the underlying JD dynamics from trajectory data and then discovers neural symmetry generators by enforcing the corresponding continuous and finite-jump invariance conditions. This formulation highlights a fundamental distinction from conventional diffusion-SDE symmetry: invariance of the continuous stochastic dynamics alone does not guarantee invariance under the discrete jump mechanism. We study recovery of JD symmetry structure from data and demonstrate its utility across three settings: improving out-of-distribution generalization, discovering and exploiting structure in jump-based stochastic generative models through Generator Matching, and symmetry-aware volatility estimation for financial time series. Together, these results establish JD symmetry as a learnable structural prior for systems with intertwined continuous and discrete stochastic dynamics. To the best of our knowledge, this is the first machine-learning work to introduce the data-driven discovery and downstream use of JD Lie symmetry, opening a new direction for symmetry learning beyond purely continuous dynamics.

1 INTRODUCTION

Many stochastic systems evolve through both continuous fluctuations and abrupt discrete events. A natural description is the *jump-diffusion (JD)* stochastic differential equation

$$dX_t = f(t, X_t)\,dt + G(t, X_t)\,dW_t + J(t, X_t)\,dN_t, \tag{1}$$

where Brownian motion W_t generates continuous stochastic variation while the Poisson process N_t produces finite jumps. This combination is important whenever diffusion alone cannot represent sudden changes in the underlying dynamics. JD processes are classical models for discontinuous asset-price movements in finance (Merton, 1976; Bates, 1996), and also arise in physical stochastic systems subject to both continuous noise and discrete shocks (Daly & Porporato, 2006). They are increasingly relevant to machine learning: Neural Jump SDEs learn continuous-time dynamics with discrete events (Jia & Benson, 2019), jump diffusions have been used for trans-dimensional generative modeling (Campbell et al., 2023), neural Merton JD models have been developed for time-series prediction (Gao et al., 2025), and Generator Matching extends generative modeling to arbitrary Markov processes, including superpositions of continuous flows and jumps (Holderrieth et al., 2025). These developments place JD dynamics alongside the broader use of neural stochastic differential equations in modern machine learning (Li et al., 2020; Kidger et al., 2021).

A complementary source of structure is *symmetry*: a transformation that changes the representation of a system while preserving its underlying law. Symmetry principles play a central role in mathematics and physics and, for differential equations, Lie symmetries map solutions to other solutions of the same dynamics and can reveal invariant structure (Olver, 1993). In machine learning, encoding known symmetries has produced architectures invariant or equivariant to permutations, translations, rotations, and more general geometric transformations (Zaheer et al., 2017; Cohen & Welling, 2016;

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.

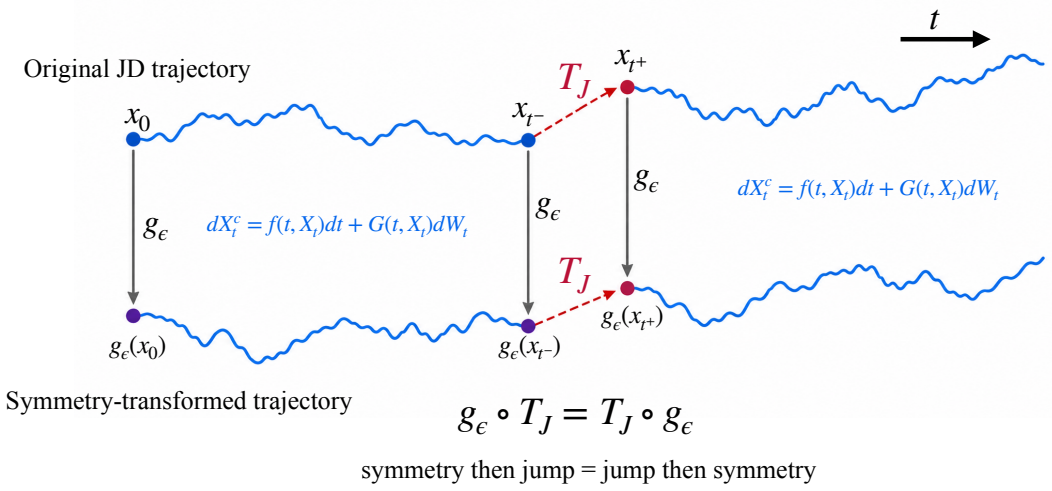


Figure 1: **Intuition for jump-diffusion symmetry.** A JD trajectory combines continuous stochastic drift–diffusion (blue) with finite jumps (red). A symmetry transformation g_ε maps the trajectory to another trajectory of the same JD dynamics, preserving both components. In particular, compatibility with the discrete dynamics requires the symmetry transformation and finite jump map T_J to commute, $g_\varepsilon \circ T_J = T_J \circ g_\varepsilon$.

Thomas et al., 2018; Fuchs et al., 2020; Satorras et al., 2021), forming a central theme of geometric deep learning (Bronstein et al., 2021). When the relevant transformation is unknown, recent work instead seeks to *discover* symmetry directly from data or learned dynamics (Dehmamy et al., 2021; Gabel et al., 2023; Forestano et al., 2023; Lishkova et al., 2023; Hou et al., 2024). This question extends naturally to stochastic dynamics: Lie-point symmetry theory has been developed for Itô SDEs (Gaeta & Spadaro, 2017; Gaeta & Lunini, 2018), and LIESTONET recently introduced a data-driven framework for discovering such symmetries from stochastic trajectories (Liu et al., 2026).

The jump component makes symmetry discovery qualitatively different. For continuous diffusion dynamics, symmetry is characterized locally through the drift, diffusion, and derivatives of the infinitesimal transformation. In a JD process, the transformation must additionally respect the finite displacement

$$T_J(t, x) = x + J(t, x), \tag{2}$$

so that applying the symmetry before or after a jump remains compatible. The JD point-symmetry theory of Nass and Fredericks makes this additional finite, nonlocal invariance condition explicit (Nass & Fredericks, 2016; 2017). Consequently, a transformation can be a valid symmetry of the continuous SDE while *not* being a symmetry of the full JD dynamics (Figure 1). This distinction becomes especially important as machine-learning models increasingly combine continuous evolution with learned jumps (Jia & Benson, 2019; Campbell et al., 2023; Holderrieth et al., 2025): imposing only continuous-dynamics symmetry can preserve the evolution between events while failing to preserve the events themselves.

JumpLie. In this work, we introduce JUMPLIE, an end-to-end framework for discovering and exploiting JD Lie symmetries directly from data. Building on the point-symmetry theory of Nass and Fredericks, JUMPLIE first learns the underlying stochastic dynamics and then learns neural infinitesimal generators constrained by both continuous and finite-jump invariance. We make three main contributions:

- **Data-driven JD symmetry discovery.** We recover analytic JD symmetry algebras from trajectory data, including settings with latent jump events, and compare against LIESTONET to show how the finite-jump constraint distinguishes genuine JD symmetries from transformations compatible only with the continuous SDE.

- **Symmetry for generalization.** We use discovered JD generators as structural inductive biases and evaluate their effect on data-efficient and out-of-distribution prediction.
- **High-dimensional and real-world applications.** We demonstrate scalability on a jump-based Generator Matching model for CIFAR-10, where discovered symmetries provide training-free generative edits, and develop a symmetry-aware JD volatility estimator for financial time series.

To the best of our knowledge, this is the first machine-learning work to introduce the data-driven discovery and downstream use of JD Lie symmetry, extending symmetry learning to stochastic dynamics governed jointly by continuous evolution and discrete events.

2 JUMP-DIFFUSION SYMMETRY

Jump-diffusion dynamics. We consider the Itô jump-diffusion

$$dX_t = f(t, X_t)\,dt + G(t, X_t)\,dW_t + J(t, X_t)\,dN_t, \tag{3}$$

where $X_t \in \mathbb{R}^d$, f is the drift, $G \in \mathbb{R}^{d\times m}$ the Brownian diffusion coefficient, $W_t \in \mathbb{R}^m$ Brownian motion, and N_t a scalar Poisson process with rate $\lambda > 0$. A jump applies the finite displacement $J(t, x)$, i.e. $T_J(t, x) = x + J(t, x)$ (Nass & Fredericks, 2016; 2017).

JD symmetry and Lie algebra. A Lie point symmetry is a smooth one-parameter transformation g_ε that maps solutions of Eq. equation 3 to solutions of the same JD dynamics. It must preserve both the continuous drift–diffusion evolution and the finite jump mechanism; schematically,

$$g_\varepsilon \circ T_J = T_J \circ g_\varepsilon,$$

as illustrated in Figure 1. Its infinitesimal generator is

$$H = \tau(t, x)\partial_t + \sum_{i=1}^{d} \xi^i(t, x)\partial_{x_i},$$

where τ and ξ describe the local time and state transformation. Admitted generators form a Lie algebra: for a basis $\{H_a\}_{a=1}^K$, $[H_a, H_b] = \sum_{c=1}^K C_{ab}^c H_c$, with structure constants C_{ab}^c. This closure property motivates the Lie-bracket objective used by JUMPLIE.

Determining equations. For a smooth function $F(t, x)$, define

$$\Gamma(F)^j = \partial_t F^j + f^i \partial_{x_i} F^j + \frac{1}{2}\sum_{k=1}^{m} G^{ik} G^{\ell k} \partial_{x_i x_\ell} F^j,$$

$$\Gamma_k^*(F)^j = G^{ik}\partial_{x_i} F^j, \;\; \Gamma^{**}(F)^j = F^j(t, x + J) - F^j(t, x).$$

Here Γ and Γ^* encode the continuous Itô dynamics, while Γ^{**} captures the finite jump. Under the temporal conditions $\Gamma(\tau) = 0,\ \Gamma_k^*(\tau) = 0,\ \Gamma^{**}(\tau) = 0$, the JD determining equations reduce to

$$R_f = H(f) - \Gamma(\xi) = 0,\ R_G = H(G) - \Gamma^*(\xi) = 0,\ R_J = H(J) - \Gamma^{**}(\xi) = 0. \tag{4}$$

The first two conditions enforce symmetry of the continuous SDE, while R_J enforces compatibility with the finite jump (Nass & Fredericks, 2016; 2017). These residuals provide the determining-equation learning signal used by JUMPLIE.

3 JUMPLIE

Figure 2 summarizes JUMPLIE. Starting from discrete stochastic trajectories, we first learn a differentiable neural surrogate of the underlying JD process. We then freeze this surrogate and learn a collection of neural vector fields whose generators satisfy the JD determining equations, remain nontrivial and independent, and close under the Lie bracket. The resulting generators can then be used as structural priors in downstream learning tasks.

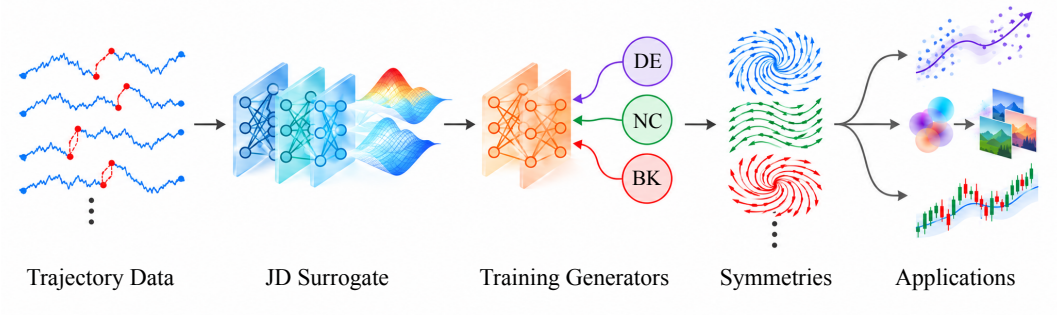


Figure 2: **Overview of the JumpLie pipeline.** From stochastic trajectories, we fit and freeze a neural JD surrogate, then learn infinitesimal symmetry generators using determining-equation (DE), non-collapse (NC), and Lie-bracket (BK) objectives. The discovered symmetries are subsequently used in downstream generalization, generation, and finance tasks.

3.1 Learning a neural jump-diffusion surrogate

Suppose we observe trajectories $\{x_n^{(r)}\}$, where $x_n^{(r)} \in \mathbb{R}^d$ is the state of trajectory r at time t_n, and define $\Delta x_n = x_{n+1} - x_n$ and $\Delta t = t_{n+1} - t_n$. We parameterize the three state-dependent coefficients of the JD model by separate neural networks,

$$f_\theta(t,x) \in \mathbb{R}^d, \qquad G_\theta(t,x) \in \mathbb{R}^{d\times m}, \qquad J_\theta(t,x) \in \mathbb{R}^d,$$

corresponding respectively to drift, Brownian diffusion, and finite jump displacement. The Poisson rate $\lambda_\theta > 0$ is represented by a separate learned positive scalar rather than another network. Thus the learned surrogate is

$$dX_t = f_\theta(t,X_t)\,dt + G_\theta(t,X_t)\,dW_t + J_\theta(t,X_t)\,dN_t.$$

When the jump count $I_n = N_{t_{n+1}} - N_{t_n}$ is observed, as in our primary recovery benchmark, the one-step conditional mean and covariance are

$$\mu_n = f_\theta(t_n,x_n)\Delta t + I_n J_\theta(t_n,x_n), \qquad \Sigma_n = G_\theta G_\theta^\top(t_n,x_n)\Delta t + \sigma_{\text{obs}}^2 I_d,$$

where $\sigma_{\text{obs}} > 0$ is a small numerical observation scale and I_d is the $d \times d$ identity. For intervals containing at most one jump, we minimize the Gaussian transition negative log-likelihood together with the exact Poisson count likelihood,

$$\mathcal{L}_{\text{trans}} = \frac{1}{2}\,\mathbb{E}_n\left[(\Delta x_n - \mu_n)^\top \Sigma_n^{-1}(\Delta x_n - \mu_n) + \log\det\Sigma_n + d\log(2\pi)\right],$$
$$\mathcal{L}_N = -\mathbb{E}_n \log \text{Pois}(I_n \mid \lambda_\theta \Delta t), \quad \mathcal{L}_{\text{surr}} = \mathcal{L}_{\text{trans}} + \mathcal{L}_N.$$

The best validation checkpoint $(\widehat{f},\widehat{G},\widehat{J},\widehat{\lambda})$ is then frozen before symmetry learning. In the latent-jump setting, where I_n is unavailable, we instead marginalize over zero- and one-jump transitions using

$$-\mathbb{E}_n \log[(1-\pi_\theta)p_0(\Delta x_n) + \pi_\theta p_1(\Delta x_n)], \qquad \pi_\theta = 1 - e^{-\lambda_\theta \Delta t},$$

where p_0 and p_1 are the corresponding Gaussian no-jump and one-jump transition densities.

3.2 Learning symmetry generators

With the surrogate frozen, we parameterize K candidate infinitesimal generators. Each generator is represented by a neural network

$$v_{\psi_a}(t,x) = \big(\tau_a(t,x), \xi_a(t,x)\big) \in \mathbb{R}^{d+1}, \qquad a = 1,\dots,K,$$

and therefore defines

$$H_a = \tau_a(t,x)\partial_t + \sum_{i=1}^{d} \xi_a^i(t,x)\partial_{x_i}.$$

Thus both the temporal component τ_a and all components of the spatial vector field ξ_a are learned directly by the generator network. Training is performed at collocation points $z = (t,x)$ sampled from the region covered by the frozen surrogate and its neighborhood.

Determining-equation loss (DE). For generator H_a, let $R_f^{(a)}, R_G^{(a)}, R_J^{(a)}$ denote the drift, diffusion, and finite-jump residuals defined in Section 2, and let $\Gamma^*(\tau_a)$, $\Gamma^{**}(\tau_a)$, and $\Gamma(\tau_a)$ denote the accompanying temporal invariance conditions. We minimize

$$\mathcal{L}_{\mathrm{DE}} = \mathbb{E}_{a,z}\Big[\|R_f^{(a)}(z)/s_f\|_2^2 + \|R_G^{(a)}(z)/s_G\|_F^2 + \|R_J^{(a)}(z)/s_J\|_2^2 + \|\Gamma^*(\tau_a)(z)\|_2^2 + |\Gamma^{**}(\tau_a)(z)|^2 + |\Gamma(\tau_a)(z)|^2\Big],$$

where s_f, s_G, s_J are fixed root-mean-square scales of the frozen drift, diffusion, and jump coefficients. This loss makes each learned vector field an infinitesimal symmetry of the complete JD dynamics, including the finite jump.

Non-collapse loss (NC). The determining equations admit the trivial zero vector field, and multiple generator networks may also converge to the same direction. To prevent both failure modes, define the generator Gram matrix

$$M_{ab} = \mathbb{E}_z\left[v_{\psi_a}(z)^\top v_{\psi_b}(z)\right], \qquad a, b = 1, \ldots, K.$$

We use

$$\mathcal{L}_{\mathrm{NC}} = \beta_{\mathrm{norm}} \sum_a (M_{aa} - 1)^2 + \beta_{\mathrm{ind}} \sum_{a \neq b} M_{ab}^2,$$

where $\beta_{\mathrm{norm}}, \beta_{\mathrm{ind}} \geq 0$ control normalization and independence. The first term prevents generators from collapsing to zero, while the second discourages redundant generators from spanning the same direction.

Lie-bracket loss (BK). A set of symmetry generators should form a Lie algebra, so their pairwise commutators must remain in their span. We jointly learn constant structure coefficients C_{ab}^c and minimize

$$\mathcal{L}_{\mathrm{BK}} = \frac{1}{|\mathcal{P}|}\mathbb{E}_z \sum_{(a,b)\in\mathcal{P}} \left\| [H_a, H_b](z) - \sum_{c=1}^{K} C_{ab}^c H_c(z) \right\|_2^2, \qquad \mathcal{P} = \{(a,b) : a < b\}.$$

Here $[H_a, H_b]$ is the usual vector-field commutator. This objective encourages the learned generators to recover a coherent finite-dimensional Lie algebra rather than a collection of unrelated symmetry directions.

The complete generator objective is therefore

$$\mathcal{L}_{\mathrm{JumpLie}} = \mathcal{L}_{\mathrm{DE}} + \mathcal{L}_{\mathrm{NC}} + \beta_{\mathrm{BK}}\mathcal{L}_{\mathrm{BK}},$$

with $\beta_{\mathrm{BK}} \geq 0$. After optimization, the learned $\{H_a\}_{a=1}^K$ are the discovered infinitesimal symmetry generators. Finite transformations can be obtained, when needed, by integrating their vector fields, and the recovered symmetry structure is subsequently used for out-of-distribution generalization, training-free transformations of jump-based generative models, and symmetry-aware financial modeling.

3.3 Relation to LieStoNet

JumpLie follows the useful two-stage principle introduced by LieStoNet - first fit and freeze a differentiable stochastic surrogate, then learn neural infinitesimal generators from symmetry constraints (Liu et al., 2026) - but differs in three important ways.

First, **a simpler objective.** JumpLie reduces LieStoNet's larger collection of algebraic and finite-step losses to three core objectives: determining equations, non-collapse, and Lie-bracket closure, i.e., $\mathcal{L}_{\mathrm{DE}}$, $\mathcal{L}_{\mathrm{NC}}$, and $\mathcal{L}_{\mathrm{BK}}$.

Second, **jump-diffusion rather than diffusion symmetry.** LieStoNet studies Wiener-driven SDE symmetry, while JumpLie targets the distinct symmetry structure of JD processes by enforcing finite-jump compatibility. This enables discovery of symmetry structure inaccessible to diffusion-only methods (Section 4.1).

Third, **greater scale and downstream use.** JumpLie extends symmetry discovery from the roughly 10-dimensional settings studied by LieStoNet to substantially higher-dimensional problems (Section 4.3), and applies the recovered symmetries to OOD generalization, generative modeling, and financial volatility estimation.

4 EXPERIMENTS

We evaluate JUMPLIE along four axes: recovery of known JD symmetries, out-of-distribution (OOD) generalization, training-free generative edits, and symmetry-aware financial volatility estimation. We begin with controlled systems whose analytic symmetry algebras are known, before moving to downstream applications.

4.1 ORACLE RECOVERY

Setup. We first test whether JUMPLIE can recover known JD symmetry structure from trajectories alone. We use three jump-diffusion systems from the analytic examples considered by (Nass & Fredericks, 2016):

$$\text{E1-A:}\quad dX_t = -kt^2 dt + \sqrt{D}\, dW_t + b\, dN_t, \qquad \mathfrak{g} = \operatorname{span}\left\{\partial_t - kt^2\partial_x,\ \partial_x\right\},$$

$$\text{E1-B:}\quad dX_t = X_t(u\, dt + \alpha\, dW_t + v\, dN_t), \qquad \mathfrak{g} = \operatorname{span}\left\{\partial_t,\ x\partial_x\right\},$$

$$\text{E1-C:}\quad \begin{cases} dX_{1,t} = X_{2,t}\, dt, \\ dX_{2,t} = -k^2 X_{2,t}\, dt + \sqrt{2k^2}\, dW_t + \alpha t\, dN_t, \end{cases} \qquad \mathfrak{g} = \operatorname{span}\left\{-\frac{e^{-k^2 t}}{k^2}\partial_{x_1} + e^{-k^2 t}\partial_{x_2},\ \partial_{x_1}\right\}.$$

The analytic equations and generators are used only to simulate trajectories and to evaluate recovery; they are not supplied to JUMPLIE. From the sampled paths and observed jump counts, JUMPLIE first learns the neural JD surrogate described in Section 3, freezes it, and then discovers the generator fields using the DE, NC, and BK objectives.

Because a Lie algebra has no unique basis, we evaluate recovery at the *subspace* level using the maximum principal angle (MPA), following LIESTONET (Liu et al., 2026). Briefly, evaluating the oracle and learned generators on the same points gives two generator matrices with orthonormal bases Q_{gt} and Q_{learn}; if σ_i are the singular values of $Q_{\mathrm{gt}}^{\top} Q_{\mathrm{learn}}$, then

$$\mathrm{MPA} = \max_i \arccos(\sigma_i).$$

We report angles in degrees, where 0° denotes identical generator subspaces and 90° complete misalignment.

Across five seeds, JUMPLIE obtains $0.548 \pm 0.030^\circ$, $0.693 \pm 0.039^\circ$, and $0.837 \pm 0.020^\circ$ for E1-A, E1-B, and E1-C, respectively. All are below one degree, showing close recovery of the analytic JD symmetry spaces. For reference, these values are also smaller than the selected-dimension principal-angle errors reported by LIESTONET on its canonical diffusion-SDE benchmarks (Liu et al., 2026). The vector fields in Figure 3 show the same agreement qualitatively; the learned basis is linearly aligned to the analytic basis only for visualization, while the MPA itself is basis invariant.

Why diffusion-only symmetry is insufficient. To isolate the role of the jump condition, we additionally run LIESTONET on the *same E1-C trajectory data*. LIESTONET models the observations as a Wiener-driven diffusion and therefore has no finite-jump compatibility condition R_J or Γ^{**}. As shown in Figure 4, its learned fields fail to recover the clean E1-C JD symmetry directions, whereas JUMPLIE recovers both. This provides a direct same-data illustration that JD symmetry is not obtained by simply applying a diffusion-symmetry method to trajectories containing jumps.

4.2 OOD GENERALIZATION

Using discovered symmetry as an inductive bias. We next ask whether the recovered generators are useful rather than merely descriptive. After symmetry discovery, we freeze the learned generators $\{\widehat{H}_a\}$ and train a fresh JD surrogate of the same architecture. The unconstrained model minimizes only the trajectory likelihood $\mathcal{L}_{\mathrm{surr}}$, whereas the symmetry-aware model minimizes

$$\mathcal{L}_{\mathrm{OOD}} = \mathcal{L}_{\mathrm{surr}} + \eta_{\mathrm{sym}} \sum_a \mathcal{L}_{\mathrm{DE}}\left(\widehat{H}_a; f_\theta, G_\theta, J_\theta\right).$$

The generators are fixed, so the second term updates only the surrogate coefficients. Intuitively, trajectory data constrain the dynamics where examples are observed, while the symmetry equations

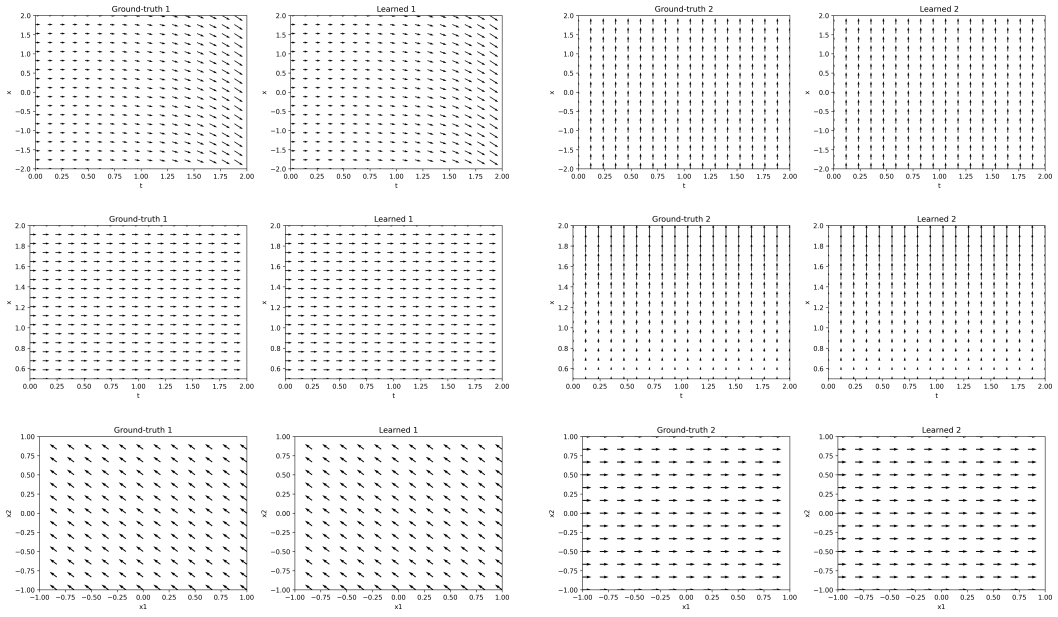


Figure 3: **Recovery of analytic jump-diffusion symmetry generators.** Ground-truth and learned vector fields are shown for the two generators of E1-A (top), E1-B (middle), and E1-C (bottom). Learned generators are aligned to the analytic basis for visualization; quantitative evaluation uses the basis-invariant maximum principal angle.

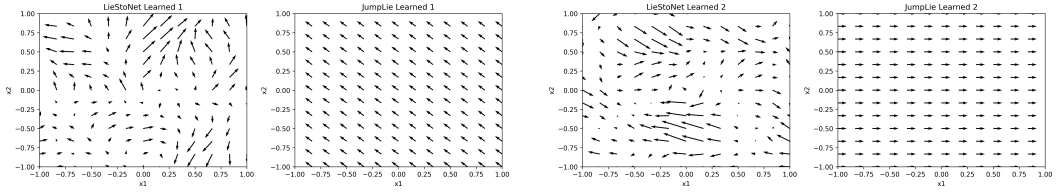


Figure 4: **Diffusion-only versus jump-diffusion symmetry discovery on E1-C.** On identical jump-diffusion trajectory data, LIESTONET produces irregular diffusion-symmetry fields, while JUMPLIE recovers the two clean JD symmetry directions.

constrain how the learned $f_\theta, G_\theta, J_\theta$ must transform away from that support. We train on a restricted state region and evaluate on symmetry-related regions outside the training distribution, using identical surrogate capacity in the two branches.

We quantify coefficient recovery by the relative RMSE

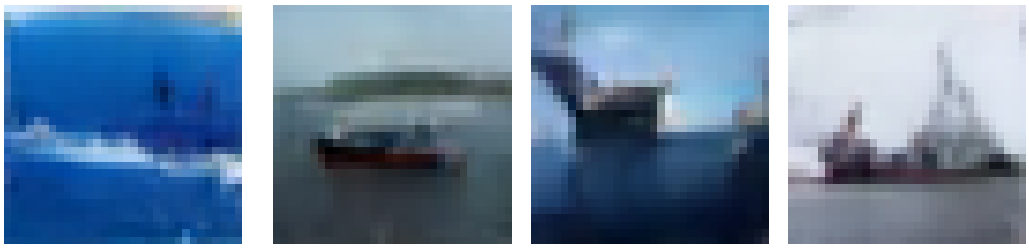
$$E_q = \frac{\sqrt{\mathbb{E}\|\widehat{q} - q\|_F^2}}{\sqrt{\mathbb{E}\|q\|_F^2} + \epsilon}, \qquad q \in \{f, G, J\},$$

where $\|\cdot\|_F$ denotes the Euclidean/Frobenius norm as appropriate and smaller values are better. Results are mean $\pm$ standard deviation over seeds.

The symmetry-aware model improves every reported coefficient across all three systems. The gains are especially large for the E1-A diffusion coefficient and the jump maps of E1-B and E1-C: relative error decreases by approximately 71%, 64%, and 72%, respectively. Improvements are smaller but still consistent for the E1-C diffusion term. Overall, the results indicate that the discovered generators act as useful structural constraints, allowing the surrogate to infer JD dynamics more accurately outside the region directly supported by training trajectories.

Table 1: **OOD coefficient recovery with discovered JD symmetry.** Relative RMSE of the learned drift f, diffusion G, and jump map J. The symmetry-aware surrogate uses frozen JUMPLIE generators; lower is better. Best values are bolded.

System	Coeff.	No symmetry	+ JumpLie symmetry
E1-A	f	$2.77\times10^{-1}\pm 9.36\times10^{-2}$	$\mathbf{1.09\times10^{-1}}\pm 1.61\times10^{-2}$
	G	$8.17\times10^{0}\pm 9.44\times10^{-1}$	$\mathbf{2.36\times10^{0}}\pm 4.78\times10^{-1}$
	J	$4.06\times10^{-1}\pm 2.45\times10^{-1}$	$\mathbf{2.11\times10^{-1}}\pm 8.49\times10^{-2}$
E1-B	f	$6.93\times10^{-1}\pm 3.90\times10^{-2}$	$\mathbf{5.27\times10^{-1}}\pm 2.35\times10^{-2}$
	G	$7.67\times10^{-1}\pm 9.44\times10^{-2}$	$\mathbf{3.69\times10^{-1}}\pm 1.99\times10^{-2}$
	J	$4.22\times10^{-1}\pm 4.65\times10^{-2}$	$\mathbf{1.50\times10^{-1}}\pm 7.15\times10^{-2}$
E1-C	f	$5.01\times10^{-1}\pm 6.97\times10^{-2}$	$\mathbf{3.76\times10^{-1}}\pm 4.03\times10^{-2}$
	G	$9.94\times10^{-1}\pm 6.07\times10^{-3}$	$\mathbf{8.41\times10^{-1}}\pm 2.33\times10^{-2}$
	J	$1.50\times10^{-1}\pm 2.55\times10^{-2}$	$\mathbf{4.17\times10^{-2}}\pm 9.48\times10^{-2}$



Base sample $x^0_{t=1}$ | Edited sample 1 | Edited sample 2 | Edited sample 3

Figure 5: **Training-free edits via learned JD symmetry directions.** Left: a base sample $x^0_{t=1}$ generated by the trained Generator Matching model. Right: three edited samples obtained by taking the intermediate state $x^0_{t=0.3}$, pushing it along three frozen learned symmetry generators for pseudo-time $\epsilon = 0.5$, and then completing the rollout with the frozen generative dynamics.

4.3 TRAINING-FREE EDITS

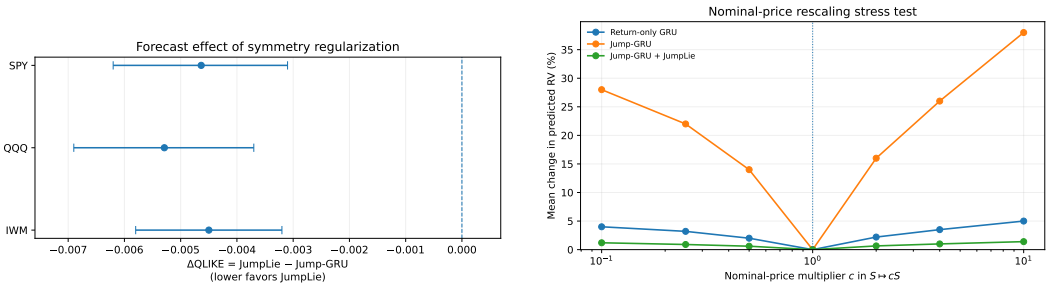
We next ask whether learned JD symmetry can be used as a controllable editing mechanism in a high-dimensional generative model. We train Generator Matching (Holderrieth et al., 2025) on the ship class of CIFAR-10, yielding a learned jump-based stochastic generative process on image states $x_t \in \mathbb{R}^{3072}$ for $32\times32\times3$ images. We then apply JUMPLIE to this frozen 3072-dimensional process and parametrize $K = 8$ neural infinitesimal symmetry generators.

After symmetry discovery, the generators are frozen and used for *training-free* edits. Let $x^0_{t=1}$ denote a generated sample at data time. We take its intermediate state $x^0_{t=0.3}$, select three of the eight learned generators, and flow each one for pseudo-time $\epsilon = 0.5$, producing $x^1_{t=0.3}, x^2_{t=0.3}, x^3_{t=0.3}$. Each perturbed state is then propagated forward by the frozen Generator Matching dynamics to obtain the final edited outputs. No retraining or test-time optimization is required.

4.4 FINANCE VOLATILITY ESTIMATION

We finally test JUMPLIE on real financial data. We use 5-minute SPY, QQQ, and IWM observations to predict next-day realized variance (RV). The relevant symmetry is particularly intuitive: changing the nominal price unit, $S \mapsto cS$, should not change the underlying volatility. In log-price coordinates this corresponds to a translation symmetry.

We first learn a JD surrogate from the training data and apply JUMPLIE to discover this structure. The discovered symmetry is then frozen and used to regularize a jump-aware GRU volatility forecaster. We compare against a classical HAR-CJ model, a Return-only GRU, and the same Jump-GRU without symmetry regularization. Full model details and numerical results are reported in Appendix A.4.



(a) Forecast improvement: lower ΔQLIKE is better.

(b) Price-rescaling robustness: lower change in RV is better.

Figure 6: **Symmetry-aware volatility forecasting.** Left: JUMPLIE improves QLIKE relative to the matched Jump-GRU baseline across SPY, QQQ, and IWM; bars show 95% block-bootstrap confidence intervals. Right: the JUMPLIE-regularized forecaster is substantially more robust to arbitrary nominal-price rescaling.

Figure 6 summarizes the two main results. **Left:** we report $\Delta\text{QLIKE} = \text{QLIKE}_{\text{JumpLie}} - \text{QLIKE}_{\text{Jump-GRU}}$, where QLIKE measures volatility-forecast error and lower is better. Negative values therefore favor JUMPLIE; all three assets show a consistent improvement over the matched Jump-GRU baseline. **Right:** we explicitly rescale the input prices by $S \mapsto cS$ and measure how much the predicted RV changes. A perfectly scale-invariant model would remain at zero. The unconstrained Jump-GRU is highly sensitive to the arbitrary nominal price scale, whereas the JUMPLIE-regularized model remains close to invariant across large rescalings.

5 CONCLUSION

We introduced JUMPLIE, a data-driven framework for discovering and exploiting Lie symmetries of jump-diffusion dynamics. The key distinction from diffusion-only symmetry learning is that a valid JD symmetry must preserve not only the continuous drift–diffusion dynamics but also the finite jump mechanism. JUMPLIE learns a neural JD surrogate from trajectories and then discovers a symmetry Lie algebra by enforcing determining-equation, non-collapse, and Lie-bracket constraints.

Across controlled and applied settings, we showed that these learned symmetries can recover known JD structure from data, improve out-of-distribution dynamics learning, scale to high-dimensional stochastic generative models where they enable training-free edits, and provide useful structural regularization for financial volatility forecasting. These results suggest that JD symmetry can serve not only as a descriptive property of stochastic dynamics, but as a practical inductive bias for learning systems that combine continuous evolution with discrete events. Future work can extend this framework to richer jump mechanisms, broader symmetry classes, and other high-dimensional event-driven systems.

AI USE STATEMENT

In this work, we used generative AI tools for [assist with translation, clean and reformat dataset, and interpreting results.] We have not used generative AI tools for [generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods], and [support qualitative and thematic data analysis] are not applicable to this work. Additionally, we used generative AI tools for [create or edit software code, summarize or analyse existing literature, edit a research paper to improve readability, format references]. We have reviewed all AI-assisted work. We checked LLM-generated texts and revised them, and LLM-generated code was verified and tested for correctness by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

To support reproducibility, we provide the complete experimental code used in this work as supplementary material. Implementation details, model architectures, optimization settings, data-processing procedures, and evaluation protocols are documented in Appendix A. Together, the appendix and supplementary code provide the information needed to reproduce the reported experiments and figures.

References

David S. Bates. Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies*, 9:69–107, 1996.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Andrew Campbell, William Harvey, Christian Weilbach, Valentin De Bortoli, Tom Rainforth, and Arnaud Doucet. Trans-dimensional generative modeling via jump diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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

Edoardo Daly and Amilcare Porporato. Probabilistic dynamics of some jump-diffusion systems. *Physical Review E*, 73:026108, 2006.

Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2503–2515, 2021.

Roy T. Forestano, Konstantin T. Matchev, Katia Matcheva, Alexander Roman, Eyup Unlu, and Sarunas Verner. Deep learning symmetries and their lie groups, algebras, and subalgebras from first principles. *arXiv preprint arXiv:2301.05638*, 2023.

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.

Alex Gabel, Victoria Klein, Riccardo Valperga, Jeroen S. W. Lamb, Kevin Webster, Rick Quax, and Efstratios Gavves. Learning lie group symmetry transformations with neural networks. *arXiv preprint arXiv:2307.01583*, 2023.

Giuseppe Gaeta and Claudia Lunini. On lie-point symmetries for itô stochastic differential equations. *Journal of Nonlinear Mathematical Physics*, 25:262–289, 2018.

Giuseppe Gaeta and Francesco Spadaro. Random lie-point symmetries of stochastic differential equations. *Journal of Mathematical Physics*, 58:053503, 2017.

Yuanpei Gao, Qi Yan, Yan Leng, and Renjie Liao. Neural mjd: Neural non-stationary merton jump diffusion for time series prediction. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Peter Holderrieth, Marton Havasi, Jason Yim, Neta Shaul, Itai Gat, Tommi Jaakkola, Brian Karrer, Ricky T. Q. Chen, and Yaron Lipman. Generator matching: Generative modeling with arbitrary markov processes. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 52153–52219, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/819aaee144cb40e887a4aa9e781b1547-Paper-Conference.pdf.

Wanda Hou, Molan Li, and Yi-Zhuang You. Machine learning symmetry discovery for classical mechanics. *arXiv preprint arXiv:2412.14632*, 2024.

Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Patrick Kidger, James Foster, Xuechen Li, and Terry Lyons. Neural sdes as infinite-dimensional gans. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 5453–5463, 2021.

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pp. 3870–3882, 2020.

Yana Lishkova, Paul Scherer, Steffen Ridderbusch, Mateja Jamnik, Pietro Liò, Sina Ober-Blőbaum, and Christian Offen. Discrete lagrangian neural networks with automatic symmetry discovery. *IFAC-PapersOnLine*, 56, 2023.

Shida Liu, Abhishek Gupta, Sumit Sinha, and L. Mahadevan. LieStoNet: Learning lie symmetries from spatiotemporal data for stochastic dynamical systems. *arXiv preprint arXiv:2608.01582*, 2026.

Robert C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3:125–144, 1976.

Aminu M. Nass and E. Fredericks. Symmetry of jump-diffusion stochastic differential equations. *Global and Stochastic Analysis*, 3(1):11–23, 2016.

Aminu M. Nass and E. Fredericks. W-symmetries of jump-diffusion itô stochastic differential equations. *Nonlinear Dynamics*, 90:2869–2877, 2017.

Peter J. Olver. *Applications of Lie Groups to Differential Equations*. Springer, 2 edition, 1993.

Víctor García Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 9323–9332, 2021.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. In *Advances in Neural Information Processing Systems*, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, pp. 3391–3401, 2017.

A Implementation Details

This appendix provides the implementation details omitted from the main text. Unless stated otherwise, all simulation, differentiation, optimization, and evaluation are implemented in JAX and compiled with XLA. PyTorch is used only to define and initialize the low-dimensional MLP and finance GRU parameterizations before converting their parameters to JAX arrays. We use single precision for the reported experiments and base random seed 17. All reported multi-seed results use independently initialized runs.

A.1 JumpLie implementation

Neural parameterization. For the low-dimensional experiments, the drift f_θ, diffusion G_θ, and jump displacement J_θ are represented by separate MLPs with four hidden layers of width 128, $\tanh$ activations, and linear output layers. Inputs t and x are standardized using statistics computed from the training set. We use Xavier initialization with zero biases. The Poisson rate is parameterized as $\lambda_\theta = \mathrm{softplus}(\ell_\theta) + 10^{-8}$. For one-dimensional experiments requiring positive diffusion, $G_\theta = \mathrm{softplus}(\widetilde{G}_\theta) + 10^{-5}$. Each symmetry generator uses the same 4×128 MLP architecture and outputs

$$h_{\psi_a}(t, x) = \big(\tau_a(t, x), \xi_a(t, x)\big) \in \mathbb{R}^{1+d}.$$

Determining equations used in code. For numerical robustness, the implementation evaluates the full determining equations before using the constraint $\Gamma(\tau) = 0$. In particular,

$$R_f = f\,\Gamma(\tau) + \frac{\lambda J}{2}\Gamma(\tau) + H(f) - \Gamma(\xi),$$
$$R_G = \frac{G}{2}\Gamma(\tau) + H(G) - \Gamma^*(\xi),$$
$$R_J = \frac{J}{2}\Gamma(\tau) + H(J) - \Gamma^{**}(\xi),$$

together with

$$\Gamma^*(\tau) = 0, \qquad \Gamma^{**}(\tau) = 0, \qquad \Gamma(\tau) = 0.$$

Thus the reduced equations displayed in the main text are mathematically equivalent to, but slightly simpler than, the residuals evaluated in the code. All required first- and second-order derivatives are computed by automatic differentiation.

Collocation and normalization. For low-dimensional discovery, every collocation batch contains equal numbers of (i) space–time points sampled from the training trajectories and (ii) off-trajectory points. For the latter, t is sampled uniformly from the observation interval and each state component uniformly from the central 99% training-data box, defined by the 0.5% and 99.5% empirical quantiles. The three determining residual families are normalized by fixed RMS coefficient scales

$$s_f = \sqrt{\mathbb{E}\|f\|^2} + 10^{-8}, \quad s_G = \sqrt{\mathbb{E}\|G\|_F^2} + 10^{-8}, \quad s_J = \sqrt{\mathbb{E}\|J\|^2} + 10^{-8},$$

computed once from a separate collocation set.

We optimize the determining-equation, non-collapse, and Lie-bracket losses jointly. The structure constants C_{ab}^c are trainable scalars and are optimized together with the generator networks. We do not add separate antisymmetry or Jacobi losses, since these properties follow from the exact vector-field Lie bracket used in the implementation.

Optimization. We use Adam with $(\beta_1, \beta_2) = (0.9, 0.999)$ and numerical constant 10^{-8}. The neural JD surrogate is selected by validation negative log-likelihood and is frozen before symmetry discovery. The generator checkpoint is likewise selected on a fixed held-out collocation set.

A.2 Oracle recovery and OOD generalization

Synthetic trajectories. For each of E1-A, E1-B, and E1-C, we simulate 5000 training, 1000 validation, and 1000 test trajectories on $t \in [0, 2]$, observed at spacing $\Delta t = 0.01$. The parameters used in the three systems are

$$\begin{aligned}
&\text{E1-A:} \quad k = 0.5, \quad D = 0.4, \quad b = 0.75, \quad \lambda = 0.8, \qquad X_0 \sim \mathrm{Unif}[-2, 2],\\
&\text{E1-B:} \quad u = -0.2, \quad \alpha = 0.35, \quad v = 0.25, \quad \lambda = 0.7, \quad X_0 \sim \mathrm{Unif}[0.5, 2],\\
&\text{E1-C:} \quad k = 1, \quad \alpha = 0.4, \quad \lambda = 0.6, \qquad\qquad\qquad (X_{1,0}, X_{2,0}) \sim \mathrm{Unif}([-1, 1]^2).
\end{aligned}$$

E1-A and E1-B are simulated directly at the observation spacing; E1-C uses the jump-diffusion simulator before retaining observations at the same $\Delta t = 0.01$. Exact Poisson jump counts are retained for each observation interval. All three systems have a two-dimensional target symmetry algebra, so we use $K = 2$ candidate generators.

Surrogate training. The observed-jump surrogate uses the likelihood described in Section 3. We add isotropic covariance $\sigma_{\mathrm{obs}}^2 I$ with $\sigma_{\mathrm{obs}} = 10^{-3}$ to the transition covariance. The Gaussian transition term is evaluated on intervals containing zero or one jump, while the exact Poisson count likelihood uses all intervals. The learned rate is initialized at $\lambda_\theta = 0.5$.

For symmetry discovery we use

$$\beta_{\mathrm{norm}} = \beta_{\mathrm{ind}} = \beta_{\mathrm{BK}} = 1.$$

Before training on the neural surrogate, the determining-equation implementation is unit-tested by inserting the analytic coefficients and analytic generators and verifying that all residual families vanish numerically.

Table 2: Low-dimensional training configuration used for E1-A–C.

	JD surrogate	**JumpLie**
Architecture	4×128 MLP	4×128 MLP
Optimizer	Adam	Adam
Learning rate	3×10^{-4}	1×10^{-4}
Batch size	512	2048 collocation points
Maximum updates	20,000	30,000
Early-stop patience	1,500	3,000
Validation points	–	8192
Scale-estimation points	–	8192
Seeds	5	5

Recovery metrics. For each seed, analytic and learned generators are evaluated on 5000 common space–time points. Their values are flattened into matrices whose columns are generator fields, orthonormalized by QR decomposition, and compared by the principal angles obtained from the singular values of the cross-Gram matrix. The maximum principal angle reported in the main text is therefore independent of the particular basis learned by the networks. For visualization only, we solve the least-squares change of basis that aligns the learned fields to the analytic basis; this alignment is not used in the quantitative metric.

LieStoNet comparison. For the E1-C comparison, LIESTONET is trained on the same simulated trajectory observations used by JUMPLIE. Its diffusion-SDE symmetry conditions do not include the finite-jump residual R_J or the nonlocal operator Γ^{**}. The comparison therefore isolates the effect of modeling the finite jump rather than changing the underlying observations.

OOD retraining. After symmetry discovery, the learned generators are detached and kept fixed. For every seed, a fresh surrogate with the same 4×128 architecture is trained either with the original surrogate likelihood alone or with

$$\mathcal{L}_{\text{surr}} + \eta_{\text{sym}} \mathcal{L}_{\text{DE}}^{\text{frozen}}, \qquad \eta_{\text{sym}} = 0.1.$$

Here $\mathcal{L}_{\text{DE}}^{\text{frozen}}$ is evaluated using the learned generators while gradients propagate only through the new $f_\theta, G_\theta, J_\theta, \lambda_\theta$. We use 512 independently sampled symmetry collocation points per update. Baseline and symmetry-aware models use identical architecture, optimization settings, data, and initialization for each matched run; no finite-orbit data augmentation is used. Evaluation is performed on symmetry-related states outside the region used to fit the fresh surrogate.

A.3 HIGH-DIMENSIONAL GENERATOR MATCHING EXPERIMENT

Dataset. We restrict CIFAR-10 to the ship class. The 5000 official training ships are deterministically divided into 4500 training and 500 validation images; the 1000 official test ships are kept untouched for final evaluation. Images are $32 \times 32 \times 3$, giving state dimension $d = 3072$, and are linearly scaled to $[-1, 1]$.

Generator Matching. We implement the continuous-flow plus finite-jump construction of Generator Matching (Holderrieth et al., 2025). For a target image z and Gaussian sample e, the conditional probability path is

$$x_t = tz + (1-t)e, \qquad t \sim \text{Unif}[0.02, 0.98],$$

with conditional flow target $z - e$. The frozen image model is a time-conditioned convolutional U-Net with base width 64 and time-embedding dimension 96. It predicts, for each RGB scalar coordinate, (i) the continuous velocity, (ii) a positive jump rate, and (iii) categorical logits over 24 possible jump destinations uniformly spaced in $[-1, 1]$. Flow and jump Generator-Matching losses receive equal weight.

The Generator Matching model is trained with Adam at learning rate 2×10^{-4}, batch size 128, for at most 60,000 updates. Validation is performed every 250 updates with early-stop patience 7500. The

jump-rate output bias is initialized to -5 to avoid an artificially dense initial jump process. Sampling uses 100 time steps and the Markov superposition weights

$$\alpha_{\text{flow}} = 0.65, \qquad \alpha_{\text{jump}} = 0.35.$$

The trained generative process is frozen before symmetry discovery.

High-dimensional JumpLie. For the final editing experiment we parameterize $K = 8$ candidate spatial generators

$$H_a = \xi_a(t, x) \cdot \nabla_x, \qquad a = 1, \dots, 8,$$

using a time-conditioned convolutional U-Net with base width 24. The network outputs $3K$ channels, which are reshaped into K RGB vector fields. Thus no 3072×3072 Jacobian is explicitly materialized.

For the continuous part, we evaluate the infinitesimal commutation residual using Jacobian-vector products,

$$Du_\theta(t, x)[\xi_a] - \partial_t \xi_a(t, x) - D\xi_a(t, x)[u_\theta(t, x)].$$

For the discrete part, we use the *exact frozen categorical jump operator* rather than a continuous relaxation. If $\lambda_j(t, x)$ and $p_{j,b}(t, x)$ denote the learned rate and categorical destination probabilities for scalar image coordinate j, then

$$(\mathcal{L}_J F)(x) = \sum_j \lambda_j(t, x) \sum_b p_{j,b}(t, x) \big[F(x^{j \to y_b}) - F(x)\big].$$

Jump symmetry is enforced through the commutator $[H_a, \mathcal{L}_J]F$. We estimate the sum over image coordinates using 8 randomly sampled scalar coordinates per state, while retaining the exact expectation over all 24 categorical destination bins. Two random smooth scalar test functions are used per update.

Symmetry training uses batches of 6 image states and learning rate 8×10^{-5}. Collocation states are drawn from a frozen trajectory bank containing 4096 training and 1024 validation states, generated from 80 trajectories with 64 temporal steps and augmented with Gaussian state perturbations of standard deviation 0.02. We use two optimization phases of at most 7500 and 5000 updates, validation every 250 updates, early-stop patience 2000, gradient norm clipping at 1, and two symmetry-discovery seeds. The regularization weights are

$$\beta_{\text{norm}} = 4, \qquad \beta_{\text{ind}} = 1, \qquad \beta_{\text{jump}} = 1, \qquad \beta_{\text{BK}} = 0.5.$$

Training-free edits. After the Generator Matching model and symmetry networks are frozen, no parameters are updated for the editing experiment. For the example in Figure 5, we retain the intermediate state $x^0_{t=0.3}$ of the base generation and independently integrate three of the learned generator fields from this state for symmetry pseudo-time $\varepsilon = 0.5$. This produces $x^1_{t=0.3}, x^2_{t=0.3}, x^3_{t=0.3}$. Each transformed state is then used as the initial condition for the remaining frozen Generator Matching rollout, producing the three displayed final images.

A.4 Finance volatility experiment

Market data and preprocessing. We use 5-minute SPY, QQQ, and IWM bars from January 2, 2018 through December 31, 2025. Timestamps are converted to New York time and only the regular trading session 09:30–16:00 is retained. We keep only complete trading days containing exactly 78 five-minute bars. For bar i, the log return is

$$r_{t,i} = \log S^{\text{close}}_{t,i} - \log S^{\text{start}}_{t,i},$$

where the start price is the first open of the day for $i = 1$ and the preceding close otherwise. Each asset is split chronologically into $60\%/20\%/20\%$ train/validation/test periods.

Daily realized variance and bipower variation are

$$RV_t = \sum_i r^2_{t,i}, \qquad BPV_t = \frac{\pi}{2} \frac{n}{n-1} \sum_{i=2}^{n} |r_{t,i}||r_{t,i-1}|,$$

with $n = 78$, and the realized jump component is $JV_t = \max(RV_t - BPV_t, 0)$. We additionally compute the standard BNS jump diagnostic using a one-sided 5% threshold.

Real-data symmetry discovery. Symmetry discovery uses only the training portion of each asset. Each five-minute observation is treated as a one-dimensional log-price transition, with

$$x = Y = \log S, \qquad \Delta x = r.$$

Because individual empirical jump events are latent, the neural surrogate is fit using the zero/one-jump mixture likelihood described in Section 3. The real market is therefore projected onto the finite-Poisson-jump model covered by the JD symmetry equations; this approximation is used for symmetry discovery and does not assume that true market jump sizes are deterministic.

The surrogate again uses 4×128 $\tanh$ MLPs, Adam learning rate 3×10^{-4}, batch size 512, at most $20{,}000$ updates, and early-stop patience 1500. Observation jitter is 10^{-5}.

We fit $K = 2$ candidate generators although the expected valid symmetry space is one-dimensional. To avoid forcing an artificial second direction, the finance experiment replaces per-generator unit normalization by the rank-adaptive condition

$$\big(\operatorname{tr} M - 1\big)^2.$$

The second candidate can therefore collapse if the data support only one generator. Symmetry discovery uses learning rate 10^{-4}, collocation batch 2048, at most $30{,}000$ updates, and patience 3000, with

$$\beta_{\text{norm}} = 1, \qquad \beta_{\text{ind}} = 0.1, \qquad \beta_{\text{BK}} = 0.05.$$

Forecasting inputs and baselines. For each day we construct the five-dimensional feature vector

$$\Big[Y_t^{\text{close}},\ \sum_i r_{t,i},\ \log RV_t,\ \log BPV_t,\ \frac{JV_t}{RV_t + \epsilon}\Big],$$

and use the preceding 22 trading days to predict next-day realized variance. The *Return-only GRU* removes the first, nominal-price channel; the *Jump-GRU* and *Jump-GRU*+JUMPLIE receive all five channels. We additionally evaluate the classical HAR-CJ baseline and an oracle-symmetry GRU as diagnostic references.

The neural forecaster contains a GRU with hidden dimension 64, followed by a hidden layer of width 128. For each of the 78 future intraday bins, the output head predicts parameters of a continuous/jump return mixture. Models are trained with Adam using learning rate 3×10^{-4}, batch size 64, at most 200 epochs, early-stop patience 20, gradient clipping at 1, and five random seeds.

Symmetry regularization. The discovered generator is frozen before forecasting. We numerically integrate its finite action on the log-price channel and precompute a bank of 17 transformations spanning symmetry pseudo-times corresponding to log-price shifts up to $|\varepsilon| = \log 2$. During training, the Jump-GRU+JUMPLIE model is penalized if its predicted next-day variance changes under these transformations:

$$\mathcal{L}_{\text{finance}} = \mathcal{L}_{\text{jump-NLL}} + 0.1\,\mathbb{E}_{\varepsilon,\mathcal{H}}\left[\log \widehat{RV}_\theta(g_\varepsilon \mathcal{H}) - \log \widehat{RV}_\theta(\mathcal{H})\right]^2.$$

Only the price-level channel is transformed; returns and realized-variation features remain unchanged. The matched Jump-GRU uses the identical architecture and likelihood without this regularizer.

Evaluation. Our primary forecasting score is

$$\text{QLIKE}(RV, \widehat{RV}) = \frac{1}{T}\sum_t \left[\frac{RV_t}{\widehat{RV}_t} - \log\left(\frac{RV_t}{\widehat{RV}_t}\right) - 1\right].$$

For the matched comparison in Figure 6, daily QLIKE losses are first averaged across forecasting seeds, after which we form $\Delta\text{QLIKE} = \text{QLIKE}_{\text{JumpLie}} - \text{QLIKE}_{\text{Jump-GRU}}$. The 95% confidence intervals use 2000 moving-block bootstrap replicates with block length 20 trading days.

For the nominal-price stress test, the complete price history is multiplied by

$$c \in \{0.1, 0.25, 0.5, 1, 2, 4, 10\}.$$

We report the mean relative change in predicted realized variance,

$$\mathbb{E}\frac{|\widehat{RV}(cS) - \widehat{RV}(S)|}{\widehat{RV}(S) + 10^{-12}},$$

averaged over assets and forecasting seeds. A value of zero corresponds to exact nominal-price-scale invariance.