Learning Compositional Latent Structure with Vector Networks

Niclas Pokel
Institute of Neuroinformatics,
UZH / ETH Zurich
ETH AI Center
Zurich, Switzerland
npokel@ethz.ch

Benjamin F. Grewe
Institute of Neuroinformatics,
UZH / ETH Zurich
ETH AI Center
Zurich, Switzerland
bgrewe@ethz.ch

Abstract

Deep networks are powerful function approximators, but they typically store many different computations in shared weight matrices, making it difficult to selectively reuse or adapt parts of them when a familiar structure appears in novel combinations. We introduce the **Vector Network** (VN), a hierarchical recurrent architecture in which each layer replaces a fixed weight matrix with a library of reusable rank-1 weight atoms. For each input, VN minimizes a layer-local energy to infer a sparse set of active weight atoms and their coefficients, jointly constrained by bottom-up input reconstruction and top-down feedback consistency. These weight atom coefficients then compose an input-specific low-rank weight matrix for that sample. After convergence, slow learning updates only the selected weight atoms through local residual signals scaled by the inferred coefficients. We evaluate VN on four compositional benchmarks spanning 1D signals, 2D spatial decoding, N-body dynamics, and compositional MNIST. VN matches strong baselines in distribution while often achieving out-of-distribution error about an order of magnitude lower when familiar factors must be recombined in novel ways. Vector networks thus make compositional generalization a structural property of the architecture and inference process rather than a brittle byproduct of fitting many behaviors into one shared dense parameter substrate.

1 Introduction

Modern deep networks trained with error backpropagation typically rely on a shared dense parameter substrate to implement many different computations. Because gradients are aggregated at the outputs and propagated through all layers, many different behaviors are written into the same shared weight space. Over training, this can produce what we term *weight entanglement*: distinct features, skills, and behaviors become encoded in overlapping directions of the same weight matrix [15]. Improvements for one behavior tend to move parameters along directions that also affect others. Since the objective constrains only the final fit, not how internal computations are decomposed, there is no direct pressure for weights to stabilize into identifiable, reusable functional components that could later be selectively recombined, although post-hoc interpretability methods can sometimes identify responsible human-aligned units or circuits [5, 33].

This matters because, in a compositional system, one would like *addressable responsibility in weight space*: learned weight components whose contributions can be isolated, recombined into a task-specific function, and updated independently. Two observations motivate this requirement: under weight entanglement no component can be reused or updated in isolation, and without an appropriate inductive bias, latent factors are generally not recovered as reusable structure [27]. In our spatial benchmark, the cost shows up directly: the OOD/ID error ratio of a conventional

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

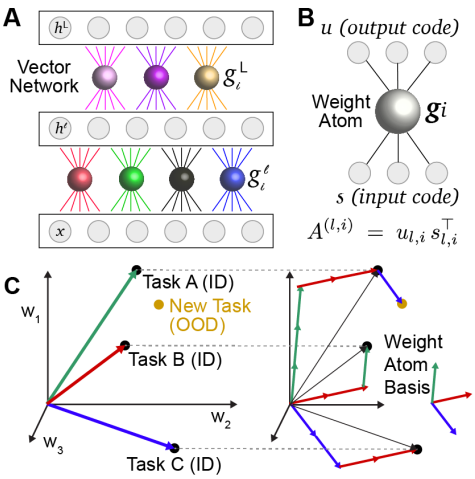


Figure 1: **Vector Networks compose input-specific function from sparse weight atoms. (A)** A VN layer replaces one dense weight matrix with reusable rank-1 weight atoms. Sparse coefficients $g_{l,i}$ are inferred to reconstruct the current input x while propagating upward information to h_l and receiving reverse reconstruction targets from above. **(B)** A weight atom is a pair of code vectors defining $A^{(l,i)} = u_{l,i} s_{l,i}^\top$, a rank-1 weight atom mapping sensing space s_l to output space u_l. The activity $g_{l,i}$ scales its contribution. **(C)** Standard networks apply a fixed monolithic matrix W, so tasks share the same parameter space. VN instead composes a few fixed weight atoms into $W_l^\star(x_l) = \sum_i g_{l,i}^\star(x_l) A^{(l,i)}$, allowing new inputs to be processed using recombinations of learned function components rather than interpolations within a fixed mapping.

CNN decoder worsens with depth (Fig. 2d). Standard deep networks do not naturally provide such persistent objects in parameter space: the problem is not only that representations become mixed, but that the corresponding transformations are stored in one shared substrate rather than in reusable units. Routing, modular, and low-rank approaches (Sec. 2) and convolutional locality [23] address parts of this problem, but do not make per-sample responsibility over reusable weight components explicit. Here we argue that if latent factors are to be recombined to explain new inputs, the learned transformation must also itself become decomposable into reusable components.

Our **Vector Network** (VN) architecture thus takes a different approach (Fig. 1A–C): it neither assigns predefined semantic roles to particular weight modules nor uses one fixed weight matrix. Instead, each layer is parameterized as a library of reusable rank-1 weight atoms (Fig. 1A), each defined by a learned code vector pair as in Fig. 1B, and for each input the model infers a sparse combination of these atoms to synthesize the effective transformation applied to that sample. *Sparse Weight-Atom Inference* optimizes a scalar coefficient for each atom under sparsity, layer-input reconstruction, and top-down feedback consistency, so that only a small active set remains. After inference has converged, the active weight atoms define the effective input-specific weight matrix (Fig. 1C). The same inferred sparse responsibility pattern concentrates slow learning on that selected subset rather than writing every sample directly into one shared full-rank matrix. In this way, vector networks make compositionality an explicit weight-space inference problem, allowing learned weight components to be selectively recombined into input-specific transformations for novel combinations of familiar structure.

Contributions.

1. **Rank-1 weight atoms as fundamental building blocks to compose network function.** Each layer maintains a dictionary of reusable rank-1 weight atoms instead of a single dense weight matrix. Network function thereby becomes *addressable*: the support of the inferred code names the atoms responsible for each sample, and this support is what slow learning acts on (Sec. 3.4; Fig. 12b–c). It also becomes *composable*: sums of learned atoms are decoded exactly and can be recovered by sparse inference (Prop. 1, Thm. 1), which the OOD results of Sec. 4 test. Each atom is moreover an explicit pair of input- and output-space vectors and thus open to inspection, which we do not pursue here.
2. **The effective weight matrix is inferred by energy minimization over weight atom coefficients.** For every sample a fast inference process minimizes a layer-local energy over weight atom coefficients under a sparsity constraint. Only after convergence do the inferred weight atoms define a tailored, input-specific $W^\star(x)$.
3. **After inference, the selected weight atoms receive Atomic-Hebb updates.** Slow learning follows the same energy locally, shaping the weight atom library through weight-atom-local residual updates scaled by inferred weight atom responsibility. Only the selected atoms are modified, which structurally limits overlap between unrelated updates.

2 Related work

Sparse coding, ICA, and dictionary learning. Classical sparse coding and ICA showed that observations can admit sparse, parts-based representations with explicit support selection [35, 36, 7, 8, 39]. This gives a clear notion of responsibility, since only the active support explains the current input and is updated. VN inherits the sparse-inference idea, but shifts it from latent-state selection to inference over reusable *weight atoms* that define the sample-specific transformation itself. Dictionary-learning methods such as K-SVD [1] learn overcomplete dictionaries of *signal* atoms whose sparse combinations reconstruct observations. VN instead learns *operator* atoms, tied pairs $(s_{l,i}, u_{l,i})$ whose sparse combination maps one layer to the next (Eq. (2)): it composes functions rather than signals. K-SVD also revises each atom jointly with its coefficients by a rank-1 SVD of the restricted residual, one atom at a time, whereas Atomic-Hebb holds the settled code fixed and updates all active atoms in parallel (Sec. 3.4). This residual×activity form is itself classical [35, 29]; VN changes the learned object, not the type of update. On function composition, the one benchmark where K-SVD applies directly, it reaches the lowest ID and Easy-OOD reconstruction error but falls behind VN on Hard-OOD reconstruction and masked prediction (App. G).

Predictive coding and energy-based inference. Predictive-coding and energy-based models also use iterative inference with local learning [38, 16, 46, 34]. VN is closely related in optimization structure: here too, fast variables are latent coefficients inferred under fixed learned dictionaries. The distinction is architectural rather than procedural. Predictive coding infers latent states within a fixed mapping, $y = \sum_j x_j w_j$ (feature selection); in VN, each inferred coefficient scales a learned rank-1 operator, $W^\star(x) = \sum_i g_i A^{(i)}$, so inference selects and composes the sample-specific operator itself (operator selection; App. A).

Routing, modularity, and object-centric approaches. Mixture-of-experts, capsules, and object-centric models address compositionality by routing among experts, part-whole representations, or object slots [22, 42, 41, 20, 18, 28]. Their selectivity operates through feedforward routing over dense modules or latent entities, whereas VN uses iterative sparse inference under an explicit energy to select shared rank-1 weight components. For mixture-of-experts, the difference also lies in what is combined: a gate mixes the *outputs* of complete dense experts, $\sum_e \pi_e(x) f_e(x)$, which interpolates between behaviors but composes no new mechanism, whereas VN sums *operators* before applying them, so $W^\star(x)$ is itself a new linear map. Each expert also remains a dense network trained by the same gradient descent that entangles dense models, so MoE partitions capacity rather than disentangling it; in VN, selection and learning share one unit, the atom.

Low-rank factorization and parameter-efficient adaptation (PEA). Low-rank factorizations and PEA methods constrain weights or updates to lower-dimensional subspaces [12, 13, 21], but they do not usually provide explicit per-sample responsibility over reusable weight components: they learn one factorization that is applied to every input. The same holds for the low-rank projections inside Transformers, whose attention weights modulate information flow but do not determine which parameters are learned (App. B). Low rank alone also does not explain VN's results: a higher-rank factorized decoder trained on rank-1 data collapses to rank 1 without OOD gain (Table 3), and sparsity-matched autoencoders underperform VN (Sec. 4.4).

3 Vector Network architecture

Next we define the core vector network architecture and weight-atom inference. We index layers by $l \in \{1, \dots, L\}$. Each layer has an input vector $x_l \in \mathbb{R}^{d_l}$, a sparse signed activity vector $g_l \in \mathbb{R}^{K_l}$ holding one coefficient per weight atom, and a top-down interface target $h_{l,\text{target}} \in \mathbb{R}^{m_l}$. Each layer stores an input reconstruction dictionary $S_l \in \mathbb{R}^{d_l \times K_l}$ and an interface dictionary $U_l \in \mathbb{R}^{m_l \times K_l}$, with columns $\{s_{l,i}\}$ and $\{u_{l,i}\}$. Implementations may use separate interface dictionaries for forward and backward messaging; below we use the tied case U_l for readability. The layers are stacked (Fig. 1A): each layer infers sparse weight atom coefficients for the current input, uses them to reconstruct the input and send a dense interface message upward, and receives a top-down target from the layer above during recurrent refinement. Dense coupling enforces $d_{l+1} = m_l$ and identifies the next-layer input with the dense message, $x_{l+1} \equiv \phi_l(U_l g_l)$, where ϕ_l is an elementwise nonlinearity,

Table 1: Principal notation. S_l, U_l (hence the atoms) are learned and fixed at test time; g_l is inferred per input.

Symbol	Meaning	Symbol	Meaning
$x_l \in \mathbb{R}^{d_l}$	input to layer l	l, L	layer index; depth
$S_l \in \mathbb{R}^{d_l \times K_l}$	input dictionary (learned)	K_l	number of atoms in layer l
$U_l \in \mathbb{R}^{m_l \times K_l}$	interface dictionary (learned)	k_l	active-atom cap (top layer)
$s_{l,i}$, $u_{l,i}$	i-th columns of S_l, U_l	E_l	layer energy, Eq. (3)
$A^{(l,i)} = u_{l,i} s_{l,i}^\top$	rank-1 weight atom	λ_l	ℓ_1 sparsity weight
$g_l \in \mathbb{R}^{K_l}$	atom coefficients (inferred)	η_l	inference step size
$W_l^\star(x_l)$	input-specific operator, Eq. (2)	$T_{\max}$, τ_E	max. sweeps; energy tolerance
$h_{l,\text{target}} \in \mathbb{R}^{m_l}$	top-down target $S_{l+1} g_{l+1}$	r_l^x, r_l^h	settled residuals, Eq. (8)
ϕ_l	nonlinearity, $x_{l+1} = \phi_l(U_l g_l)$	ρ_l^S, ρ_l^U	slow learning rates

typically identity or ReLU. Table 1 collects the notation, and Algorithm 1 summarizes one complete inference-and-learning iteration.

3.1 Rank-1 weight atoms as forward-feedback operators

In layer l, weight atom i is a paired set of unit code vectors in input and output space (Fig. 1B; maintained at unit norm by the per-column renormalization in Sec. 3.4), which together define the rank-1 atom $A^{(l,i)}$:

$$s_{l,i} \in \mathbb{R}^{d_l}, \qquad u_{l,i} \in \mathbb{R}^{m_l}, \qquad A^{(l,i)} = u_{l,i} s_{l,i}^\top. \tag{1}$$

The atoms themselves are dense; sparsity is imposed only on the coefficients g_l that select and scale them (Sec. 3.2). After inference has settled, the weight atom coefficients $g_l^\star(x_l) \in \mathbb{R}^{K_l}$ synthesize an instance-conditioned weight matrix as a short sum of these weight atoms (Fig. 1C),

$$W_l^\star(x_l) \;=\; \sum_{i=1}^{K_l} g_{l,i}^\star(x_l)\, A^{(l,i)}. \tag{2}$$

The same atom coefficients g_l serve two coupled roles: they reconstruct the current layer input as $\hat{x}_l = S_l g_l$ and, through the layer output dictionary U_l, generate the dense upward message $h_l = \phi_l(U_l g_l)$ passed to the next layer; $W_l^\star(x_l)$ is a compact description of the operator induced by this sparse code. Because each $A^{(l,i)}$ is rank 1, the rank of $W_l^\star(x_l)$ is limited by how many weight atoms are active in $g_l^\star$. Under an explicit hard top-k_l constraint, at most k_l atom coefficients are allowed to be nonzero, so the per-sample transformation has rank at most k_l regardless of the total dictionary size (see Proposition 2, Appendix E.4). Because the reconstruction map is linear in the sparse code, superpositions are preserved at the decoder level: for two codes g_a and g_b, $S_l(g_a + g_b) = S_l g_a + S_l g_b$; whether inference recovers such novel compositions depends on the conditions in App. E.1.

3.2 Layer-local energy for input reconstruction, atom sparsity, and top-down consistency

The deep VN tracks a layer-local energy in the sparse coefficients g_l. For layer l, we define the tied energy form as

$$E_l(g_l \mid x_l, h_{l,\text{target}}) = \tfrac{1}{2}\|x_l - S_l g_l\|_2^2 \;+\; \lambda_l \|g_l\|_1 \;+\; \tfrac{1}{2}\|h_{l,\text{target}} - U_l g_l\|_2^2. \tag{3}$$

Here λ_l is the layer-specific sparsity parameter; in our models, higher layers use progressively sparser, more selective codes (App. J). The energy terms have the following roles: (i) $S_l g_l$ reconstructs the current input x_l, and (ii) $U_l g_l$ remains consistent with $h_{l,\text{target}}$, the interface code supplied by deeper layers. At the top layer there is no higher layer to provide $h_{L,\text{target}}$, so the top-down term is omitted. Bottom-up and top-down are thus not separate processing stages but two consistency constraints on the same coefficient vector g_l: it must explain the input from below and agree with the target predicted from above. The top-down term arises only through hierarchical coupling and is absent in a single-layer VN. The ℓ_1 penalty induces sparsity through the proximal soft-thresholding update $\text{soft}(z, \tau) = \text{sign}(z)\max(|z| - \tau, 0)$, setting coefficients below threshold to zero.

For fixed $x_l, h_{l,\text{target}}, S_l, U_l$, E_l is a Lyapunov function of the fast inference: accepted proximal steps never increase it. In the deep stack the targets are refreshed during inference, so the summed

Algorithm 1 One VN iteration: fast inference (settle), then slow Atomic-Hebb learning; at test time only the settle runs. Plain ISTA is shown (FISTA adds momentum); slow updates are batch-averaged and applied with Adam.

Require: input x_1; dictionaries $\{S_l, U_l\}_{l=1}^{L}$; hyperparameters $\lambda_l, \eta_l, k_L, T_{\max}, \tau_E, \rho_l^S, \rho_l^U$ (Table 1)

$g_l \leftarrow 0$, $h_{l,\text{target}} \leftarrow 0$ for all l ▷ or warm start

for $t = 1, \dots, T_{\max}$ **do** ▷ fast inference: S_l, U_l fixed

  for $l = 1, \dots, L$ **do** ▷ upward pass

    $g_l \leftarrow \text{soft}\big(g_l - \eta_l \nabla f_l(g_l),\ \eta_l \lambda_l\big)$; apply Π_{k_L} if $l = L$ ▷ Eqs. (4)–(6)

    if $l < L$ **then** $x_{l+1} \leftarrow \phi_l(U_l g_l)$ ▷ Eq. (7)

  for $l = L-1, \dots, 1$ **do** ▷ downward pass

    $h_{l,\text{target}} \leftarrow S_{l+1} g_{l+1}$ ▷ Eq. (7)

    $g_l \leftarrow \text{soft}\big(g_l - \eta_l \nabla f_l(g_l),\ \eta_l \lambda_l\big)$ ▷ Eq. (5)

  if the relative decrease of $\sum_l E_l$ is below τ_E **then break** ▷ Sec. 3.3

for $l = 1, \dots, L$ **do** ▷ slow learning with $g_l^\star \leftarrow g_l$ (training only)

  $r_l^x \leftarrow x_l - S_l g_l^\star$; $r_l^h \leftarrow h_{l,\text{target}} - U_l g_l^\star$ if $l < L$ ▷ Eq. (8)

  $S_l \leftarrow S_l + \rho_l^S\, r_l^x (g_l^\star)^\top$; $U_l \leftarrow U_l + \rho_l^U\, r_l^h (g_l^\star)^\top$ if $l < L$ ▷ Eq. (9)

  renormalize the columns of S_l and U_l to unit norm

layer energies serve as per-sweep accounting rather than a global Lyapunov function. Fast inference minimizes the energy over g_l; slow learning minimizes it over the selected atoms.

3.3 Fast inference of a sparse set of rank-1 weight-atom coefficients

Inference in the deep VN proceeds by a small number of global upward and downward sweeps. At each layer, coefficient inference is a proximal-gradient step on the layer energy E_l, i.e. an ISTA-style update [37], with optional FISTA-style acceleration [6]. For layer l, we denote by $\nabla f_l(g_l)$ the gradient of the smooth part of E_l, i.e. the quadratic reconstruction and top-down consistency terms:

$$\nabla f_l(g_l) = S_l^\top (S_l g_l - x_l) + U_l^\top (U_l g_l - h_{l,\text{target}}), \tag{4}$$

where the U term is omitted when $h_{l,\text{target}}$ is not defined (top layer). With layer-specific step size $\eta_l > 0$, the soft-k update and its hard-k variant are

$$g_l \leftarrow \text{soft}(g_l - \eta_l\, \nabla f_l(g_l),\ \eta_l \lambda_l)\,, \tag{5}$$

$$g_l \leftarrow \Pi_{k_l}\Big(\text{soft}\big(g_l - \eta_l\, \nabla f_l(g_l),\ \eta_l \lambda_l\big)\Big). \tag{6}$$

Eq. (6) adds an explicit top-k_l projection Π_{k_l}, which keeps the k_l largest-magnitude coefficients when a strict cardinality budget is required. In our experiments, hidden layers use the soft-k regime, while the top layer may use a strict hard top-k budget (see Remark 5 for the design rationale). At the network level, these local coefficient updates are organized into recurrent upward and downward passes, with the downward pass acting as a reverse reconstruction path in an encoder–decoder-like settle:

$$\text{upward pass: } x_{l+1} \leftarrow h_l = \phi_l(U_l g_l), \qquad \text{downward pass: } h_{l,\text{target}} \leftarrow \hat{x}_{l+1} = S_{l+1} g_{l+1}. \tag{7}$$

Each sweep takes one update per layer on the way up and one per lower layer on the way down (Algorithm 1), where the refreshed target adds a top-down residual (App. E.5); inference stops when the relative energy improvement falls below τ_E or after $T_{\max}$ sweeps, and sweeps that increase the energy may be rejected. With η_l scaled by the inverse Lipschitz constant of the local Hessian, the soft-k proximal updates monotonically decrease the layer energy and converge to a stationary point (Theorem 3, Appendix E.3).

3.4 Slow learning updates the inferred rank-1 weight atoms

The dictionaries S_l and U_l, and hence the atoms $A^{(l,i)}$, are the learned parameters of each VN layer. They are updated during training and fixed at test time, where the only per-sample optimization is the inference of g_l (Sec. 3.3), as in classical sparse coding. Once fast inference has converged, learning updates only the selected atom vectors, i.e. the dictionary columns with nonzero coefficients, using strictly local residual$\times$atom coefficient signals. The inactive atoms are left unchanged, and inference

Table 2: Benchmarks at a glance. All models are trained on ID data only; *non-compositional shift.

Benchmark	Input → output	ID (training)	OOD (test)
Spatial	56-D bump code → 28×28 bump image	outside holdout	inside a square holdout
Functions	512-D (masked) signal → full signal	single primitives	Easy: sums; Hard: nested
N-body	22-D per-body state → 2-D acceleration	one force, n=5	2–4 forces, n=5, 4, 3 bodies
MNIST	28×28 image → same image	single digits	digit compositions; occlusion, noise*
RTS	40×40 motion frame dx → same frame	single transform	combined, larger, multi-digit motions

itself is treated as a black box that is *not* differentiated through. For layer l, we define the settled residual errors in the two spaces as

$$r_l^x = x_l - S_l g_l^\star, \qquad r_l^h = h_{l,\text{target}} - U_l g_l^\star \tag{8}$$

(when $h_{l,\text{target}}$ is defined). A direct local gradient step is then

$$S_l \leftarrow S_l + \rho_l^S \, r_l^x \, (g_l^\star)^\top, \qquad U_l \leftarrow U_l + \rho_l^U \, r_l^h \, (g_l^\star)^\top, \tag{9}$$

where $r\, g^\top$ denotes the batch-averaged outer product over a batch of size B, and ρ_l^S, ρ_l^U are local learning rates. For the top layer (no $h_{L,\text{target}}$), the interface update is skipped; in the implementation, these terms are used as local gradients for Adam, followed by renormalization. We term this rule *Atomic-Hebb*: unlike pre–post(error) Hebbian plasticity, each step combines the local residual with an inferred atom coefficient, so credit assignment is weight-atom local rather than neuron local. By Danskin's theorem, these local products are the exact implicit gradients of the settled energy (Theorem 2, App. E.2).

4 Experiments

Across the following four experiments, models are trained in distribution (ID) and evaluated on both ID and out-of-distribution (OOD) cases formed by novel combinations of previously seen factors. Because most are reconstruction tasks, we report absolute reconstruction error and how performance changes from familiar ID structure to composed OOD cases. Table 2 summarizes the inputs, outputs, and ID/OOD splits of all benchmarks; architectures and hyperparameters are given in App. J.

4.1 Spatial composition: generalizing to novel bump positions

We begin with a simple spatial reconstruction benchmark adapted from Liang et al. [25], because it isolates the weight entanglement problem in a particularly clean form. As shown in Fig. 2a, the model receives a low-dimensional code for the x- and y-position of a target bump and must reconstruct a Gaussian bump at that location on a 28×28 grid. Training is restricted to a local spatial region outside a square holdout zone, while OOD evaluation probes that unseen center region. This makes the benchmark a minimal test of whether a learned decoder can preserve a reusable spatial factorization off support, rather than only interpolating within the training region.

Our reproduction recovers the basic pattern reported by Liang et al. [25]. The spatial error maps in Fig. 2b show that standard CNN- and VAE-based decoders develop pronounced OOD error inside the holdout zone, whereas hand-factorized decoders remain more structured but less performant. Quantitatively, Fig. 2c shows the core tradeoff: the CNN exhibits a large OOD decline, whereas the hand-built rank-1 factorized decoder generalizes more predictably but remains limited in absolute reconstruction performance. Fig. 2d shows that the OOD/ID ratio of the CNN decoder worsens with depth, i.e. the deeper the network, the more strongly entangled its off-support mapping becomes.

VN combines the two desiderata. A VN decoder maintains uniformly low reconstruction error across the full spatial domain (Fig. 2b), achieves substantially lower OOD error while preserving the strongest ID performance (Fig. 2c), and even improves slightly with depth (Fig. 2d). The full VN encoder–decoder improves robustness further by discovering the factorization end to end rather than receiving it as a hand-built prior. Even in this simple setting, VN thus learns a reusable decomposition in its weight atoms that supports reconstruction in the unseen region. Extended results and ablations are in App. F.

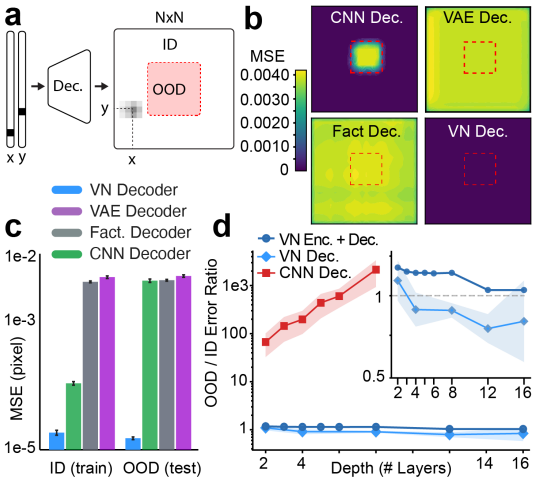


Figure 2: **Spatial composition: generalizing to novel bump positions.** **(a)** Task setup (adapted from [25]). A decoder maps (x, y) to an $N \times N$ output grid. Training covers a local ID region, while testing probes an unseen OOD region. **(b)** Spatial reconstruction error (MSE) across the grid. CNN, VAE, and factorized decoders show strong OOD error, whereas the VN decoder remains low across the full space. **(c)** Reconstruction MSE on ID and OOD regions. VN-based models achieve much lower OOD error while maintaining strong ID performance. Error bars are 95% CI. **(d)** OOD/ID error ratio vs. depth. CNN decoders degrade rapidly with depth, whereas VN remains stable and improves with depth (inset: zoom on VNs).

4.2 Function composition: generalizing to unseen signal combinations

We next test the compositional capabilities of VN by training on single primitive functions and evaluating on unseen compositions of these primitives (Fig. 3a). Each 1D signal is sampled from one of four elementary function families, $a\sin(ft+\phi)$, $a\cos(ft+\phi)$, a bounded polynomial $a(t/2\pi)^f$, or a Gaussian bump $a\,e^{-0.3f(t-\pi)^2}$, with randomized parameters. The ID task is to reconstruct signals from a single family, while the OOD sets require novel compositions of those same primitives: Easy-OOD uses cross-family sums such as $\sin(f_1t+\phi_1)+a\cos(f_2t+\phi_2)$, whereas Hard-OOD uses nested and mixed nonlinear compositions such as $\sin\big(\cos(f_1t)+f_2t/(2\pi)\big)+a\cos\big(f_3(t/(2\pi))^2\big)$. The two OOD sets differ in kind: Easy-OOD sums lie in the span of the learned primitives, so the superposition argument of Prop. 1 applies, whereas Hard-OOD nesting is not an additive superposition and lies outside the guarantees of Prop. 1 and Thm. 1. The first protocol is reconstruction.

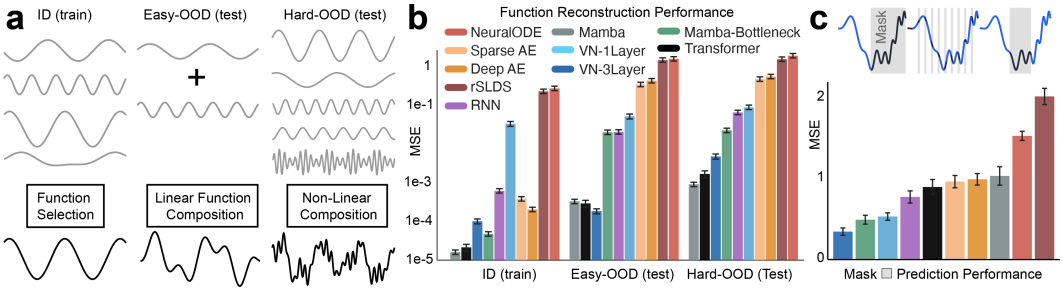


Figure 3: **Function composition: generalizing to novel signal combinations.** **(a)** Training and test structure. Models are trained on individual function families drawn from the in-distribution (ID) set. Easy-OOD tests require linear recombination of learned functions, whereas Hard-OOD tests require non-linear compositions of previously learned functions. **(b)** Reconstruction MSE for ID, Easy-OOD, and Hard-OOD conditions. **(c)** Average Hard-OOD masked prediction MSE across the four masking regimes shown above. All bar plots are sorted by mean performance; error bars are 95% CI. VN-1L/3L: 1 and 3 layer VN.

The reconstruction results in Fig. 3b show that the benchmark is nontrivial, but they do not yet cleanly separate compositional structure learning from strong sequence modeling. On Easy-OOD, VN achieves lower reconstruction error than both the Transformer and Mamba, whereas on Hard-OOD both sequence models outperform VN (Table 7). Reconstruction admits shortcut solutions: when much of the relevant signal is directly present in the input, a strong sequence model can achieve low error by preserving or continuing that structure without recovering components that can be recombined systematically, whereas VN must explain each sample through a small active set of weight atoms.

To remove this shortcut, we introduce the masked prediction variant in Fig. 3c, where part of the signal must be forecast or filled in from the learned representation. Under this evaluation, the baseline

models degrade much more strongly, whereas VN remains substantially more stable and achieves the clearest advantage, ranking first in every Hard-OOD cell (Table 8). The bottlenecked Mamba variant shows the same effect from the baseline side: it is worse than Mamba in every reconstruction regime but better in every masked cell, suggesting that constraining the representation helps precisely when the near-identity shortcut is removed. Taken together, these results suggest that VN's strength in this benchmark is not simply smoother reconstruction, but the recovery of sparse reusable latent structure that supports both reconstruction and prediction under novel function compositions. Full results are in App. G.

4.3 Force composition: generalization in n-body dynamics

We next turn to force composition in n-body dynamics, a substantially harder physics benchmark in which the relevant latent factors are the underlying force laws themselves (Fig. 4a). Models are trained only on single-force systems (gravity, drag, Lorentz, or spring) and are then evaluated by predicting per-body accelerations under two simultaneous OOD shifts: multiple forces acting together and reduced body count. This is a much stronger compositional test than interpolation alone: although instantaneous forces superpose linearly, the resulting trajectories diverge under coupled state feedback through positions and velocities, so the model must reuse learned physical mechanisms in unseen superpositions and under shifted state distributions.

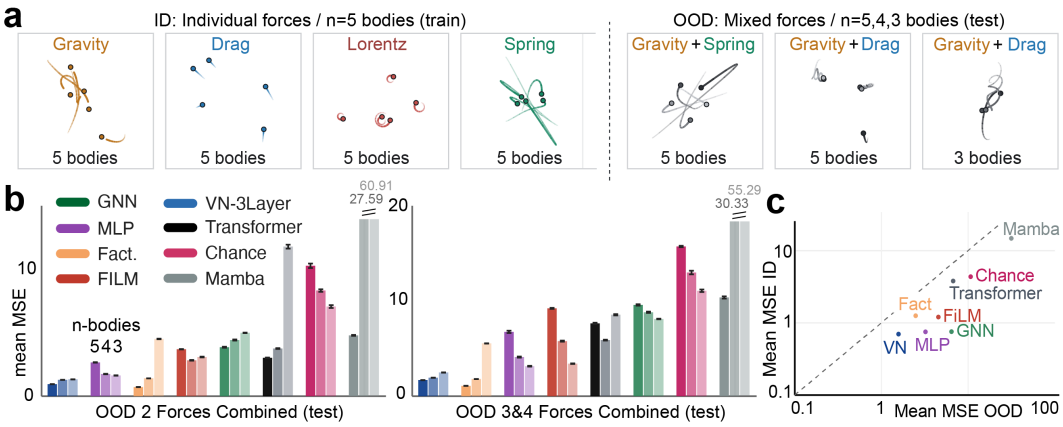


Figure 4: **Force composition: generalization in n-body dynamics.** **(a)** Task setup. Models are trained on single force fields (gravity, drag, Lorentz, spring) with $n = 5$ bodies and then evaluated out of distribution on mixed-force systems and reduced body counts ($n = 5, 4, 3$). **(b)** Mean OOD MSE under force composition. Left: mean over all two-force combinations. Right: mean over all three- and four-force combinations. Error bars are 95% CIs; within each model, lighter bars denote reduced body count ($n = 4, 3$). **(c)** Mean OOD vs. mean ID performance on a log–log scale, averaged across $n = 5, 4, 3$. Each point is one model, with ID MSE on the y-axis and OOD MSE on the x-axis. The dashed diagonal marks equal ID and OOD performance; models that degrade under distribution shift move to the right and below this line. VN remains closest to the ideal low-error, near-diagonal regime, indicating stronger robustness and compositional generalization.

The main quantitative result is shown in Fig. 4b. All models perform well on the single-force ID cases (ID errors are shown on the y-axis of Fig. 4c), but errors rise substantially once forces are combined, and rise further as evaluation moves from two-force mixtures to the harder three- and four-force regimes. The effect becomes even stronger under body dropout, shown by the lighter $n = 4$ and $n = 3$ bars. In contrast, the 3-layer VN remains consistently low across these OOD settings and retains a clear margin over MLP, GNN, Transformer, Mamba, and FiLM baselines. We also compare against a 'Factorized' control, which uses one hand-crafted subnetwork per force and combines their fitted contributions at test time, as well as a random baseline corresponding to chance-level prediction. Even against these controls, fitting the primitive regimes in isolation is not enough: what matters is whether the learned mechanisms can still be recombined when the physical system changes. For extended results see Appendix H.

Figure 4c compares mean ID and OOD error directly: several baselines achieve low ID error yet move far to the right under compositional shift, indicating that good single-force reconstruction does not imply robust transfer to mixed-force dynamics. VN remains closest to the low-error, near-diagonal regime, showing that its advantage here is not simply stronger ID fitting, but more stable reuse of learned force-law structure under increasing compositional load. In that sense, the n-body benchmark provides the clearest evidence in our suite that explicit weight-space responsibility can support mechanism-level composition in a difficult nonlinear setting.

4.4 Image composition: generalizing to novel MNIST digit compositions

To test whether the same VN capability transfers to vision, we evaluate two compositional MNIST [23] benchmarks constructed for this study (Fig. 5a–c). The first is a digit reconstruction benchmark in which the ID setting consists of single digits, while the OOD conditions require compositions such as Add/Subtract mixtures, Max projections over multiple digits, and inversion combined with other changes; two further conditions (occlusion, noise) are corruptions rather than compositions and test robustness to non-compositional shift. Appendix I reports additional representation and convergence diagnostics. The second is a dynamic Rotation/Translation/Scaling (RTS) benchmark in which single transformations define the ID setting and OOD evaluation requires stronger and more complex transformation compositions, including larger step sizes and multiple digits transformed simultaneously (Fig. 5b). VN is compared against autoencoder (AE) baselines matched in ID performance or in bottleneck sparsity, so any OOD differences cannot be explained by capacity or sparsity alone.

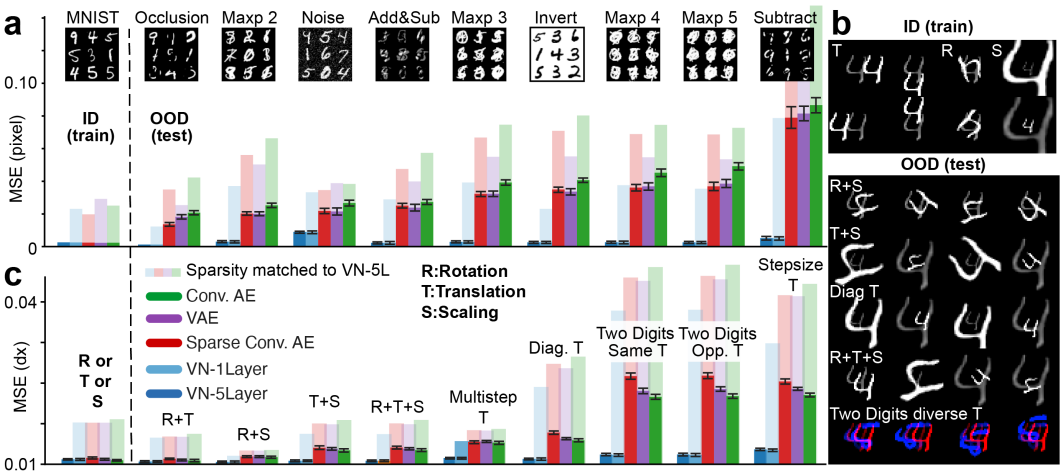


Figure 5: **Image composition: generalizing to MNIST compositions.** **(a)** VN reconstruction error on MNIST ID and OOD composition tests. **(b)** Representative Rotation/Translation/Scaling (RTS) ID training samples and OOD test compositions of different motions. **(c)** Motion reconstruction error on ID/OOD RTS transformations; faint bars denote sparsity-matched AE variants. OOD tests combine familiar transformations outside the training distribution, and VN again achieves the strongest OOD performance while remaining competitive on ID. Error bars are 95% CI.

The digit benchmark already shows a clear separation (Fig. 5a). All models perform well on single-digit ID cases, but the difference appears once the test set requires composition or corruption. Under Add/Subtract mixtures, Max projections, and structured corruptions, AE reconstruction errors increase substantially, whereas the VN remains much closer to its ID performance. In the strongest OOD conditions, VNs are roughly an order of magnitude better than the corresponding AE baselines. Sparsity-matched AEs also perform worse across both ID and OOD tests, showing that the advantage is not sparsity alone but the VN's sparse weight-atom structure and reuse.

The RTS benchmark shows the same pattern for visual transformations rather than digit identity (Fig. 5b–c). For single transformations, all models perform similarly. When the OOD setting requires larger transformation steps or simultaneous transformations of multiple digits, AE baseline errors increase steadily, while the VN degrades much more gracefully and preserves digit structure more

reliably. Thus the same advantage that appears in static digit composition also extends to a dynamic visual setting in which the relevant factors are transformations rather than image content.

5 Discussion

VN is often comparable to strong baselines in distribution, and its advantage appears when previously learned components must be recombined: there it frequently achieves substantially lower error, often by about an order of magnitude, including against Transformers and Mamba. We attribute this to making weight responsibility explicit before plasticity. VN first infers which weight atoms explain the current input and then updates only those (Atomic-Hebb); this fast-recombination / slow-storage split localizes plasticity and limits overlap between unrelated updates while preserving recomposability across inputs (see Remark 4 for the contrast to backpropagation).

The same weight-atom structure suggests several broader advantages. First, once a useful library of latent factors has been learned, new inputs can be handled by recomposition rather than by storing each factorial combination separately. This offers a natural route toward improved data efficiency, especially in the long tail, where the challenge is not simply interpolation between seen cases but reuse of familiar parts in unfamiliar combinations. In this sense, VN may scale more gracefully with the number of latent factors that must be discovered than with the number of training examples needed to cover their combinations. Explicit weight-atom responsibility may also be useful for continual learning, because it provides a direct sample-wise signal of which reusable components were important for the current explanation. A cautious extension is that some new behavior may be incorporated by refining a small subset of existing components, or where necessary adding a few new ones, rather than reworking an entire dense shared substrate. In spirit this is related to parameter-efficient schemes and their variants such as LoRA, which also restrict adaptation to a low-rank subspace, but here the relevant components are selected and later complemented by input-dependent inference rather than introduced as fixed adaptation factors [21].

Limitations. First, compositional generalization is conditional. Additive combinations of learned atoms lie in the model's span exactly (Prop. 1), but sparse inference recovers them only from a faithful, sufficiently incoherent dictionary (Thm. 1, Remark 1); recovery fails when an input needs more than k active atoms, when atoms collapse into near-duplicates, or when the settle stops early (Remark 7), and nested compositions such as Hard-OOD lie outside the guarantee. Second, inference is costly. One proximal update at layer l takes about $2K_l(d_l+m_l)$ multiply-accumulates, so T sweeps cost $O(T\sum_l K_l(d_l+m_l))$, versus $O(d_l m_l)$ for one pass through a dense layer; we use $T_{\max}=5$ (function composition, n-body), up to 15 (spatial), and up to 120 (MNIST), and masked prediction costs about five times more. Because every update evaluates the full dictionary, sparsity currently adds capacity rather than saving computation, and VN inference is slower than a feedforward pass, whereas a sparse mixture-of-experts layer executes only a few experts. Restricting late sweeps to the stabilized support would reduce the per-step cost toward $O(k_l(d_l+m_l))$, and learned approximations of iterative sparse inference [19, 30, 31], adaptive halting [17], or warm starts could shorten the settle. Third, the dictionary is sized by hand: k must cover the mechanisms active at once while staying well below $m_l d_l/(m_l+d_l)$ (Remarks 6–7; Table 4), K is chosen by validation, and growing the dictionary for new task types remains untested.

Overall, the contribution of the VN is not only in the current benchmarks, but in the design principle it makes explicit: generalization should rely on reusable weight structure rather than on continually fitting new cases into one monolithic parameter field. Conceptually, VN shifts compositionality from state routing to responsibility in weight space, which may matter most in real-world settings where familiar factors must be reused in unfamiliar combinations.

Acknowledgments and Disclosure of Funding

We thank Sander de Haan, Giuseppe Chindemi and Yassine Taoudi-Benchekroun for comments on the manuscript. This work was supported by the Swiss National Science Foundation (189251, 10003772, B.F.G.) and ETH Zurich project funding (ETH-20 19-01, B.F.G.). The authors declare no competing interests.

Author contributions

B.F.G. conceived the Vector Network architecture, Atomic-Hebb rule and developed the first multi-layer implementation. N.P. and B.F.G. jointly developed and wrote the present manuscript and the OOD experimental settings. N.P. carried out the mathematical proofs, the function-composition, spatial-decoding (blob), and n-body experiments, and contributed substantially to the experimental analysis and presentation. B.F.G. and N.P. jointly refined the implementation and the final experimental framework.

References

[1] M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006. doi: 10.1109/TSP.2006.881199.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[3] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[4] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[5] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[7] Anthony J. Bell and Terrence J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995. Neural Computation link.

[8] Anthony J. Bell and Terrence J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997. Vision Research link.

[9] Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2nd edition, 1999.

[10] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[11] John M. Danskin. *The theory of max-min and its application to weapons allocation problems*. Springer-Verlag, 1967.

[12] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2013. arXiv:1306.0543.

[13] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure in cnns for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. arXiv:1404.0736.

[14] David L. Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences (PNAS)*, 100(5):2197–2202, 2003.

[15] Nelson Elhage, Tristan Hume, Catherine Olsson, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

[16] Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010. Nature Reviews Neuroscience link.

[17] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[18] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2424–2433. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/greff19a.html`.

[19] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning (ICML)*, 2010.

[20] Geoffrey E. Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International Conference on Learning Representations (ICLR)*, 2018. ICLR link.

[21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2106.09685.

[22] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. Neural Computation link.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Yann LeCun, Sumit Chopra, Raia Hadsell, et al. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

[25] Qiyao Liang, Daoyuan Qian, Liu Ziyin, and Ila Fiete. Compositional generalization via forced rendering of disentangled latents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL `https://openreview.net/pdf/d94a41221acd289b56e4f17f3a3a6390c9c0cee9.pdf`.

[26] Timothy P. Lillicrap, Daniel Cownden, Colin J. Akerman, and Douglas B. Tweed. Random feedback weights support learning in deep neural networks. *Nature Communications*, 7:13276, 2016.

[27] Francesco Locatello, Stefan Bauer, Mario Lucic, et al. Challenging common assumptions in unsupervised learning of disentangled representations. In *International Conference on Machine Learning (ICML)*, 2019.

[28] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, et al. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[29] Julien Mairal, Francis R. Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 689–696, 2009. URL `https://doi.org/10.1145/1553374.1553463`.

[30] Joe Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In *International Conference on Machine Learning (ICML)*, 2018.

[31] Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

[32] Erkki Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

[33] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

[34] Gaspard Oliviers, Mufeng Tang, and Rafal Bogacz. Bidirectional predictive coding, 2025. URL `https://arxiv.org/abs/2505.23415`.

[35] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996. Nature link.

[36] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997. Vision Research link.

[37] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014. doi: 10.1561/2400000003.

[38] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999. Nature Neuroscience link.

[39] Christopher J. Rozell, Don H. Johnson, Richard G. Baraniuk, and Bruno A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10): 2526–2563, 2008. doi: 10.1162/neco.2008.03-07-486.

[40] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. Technical report, 01 2008.

[41] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. arXiv:1710.09829.

[42] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL `https://arxiv.org/abs/1701.06538`.

[43] Jeremias Sulam, Vardan Papyan, Yaniv Romano, and Michael Elad. Multi-layer convolutional sparse modeling: Pursuit and dictionary learning. *IEEE Transactions on Signal Processing*, 66 (15):4090–4104, 2018.

[44] Ryan J. Tibshirani. The lasso problem and uniqueness. *Electronic Journal of Statistics*, 7: 1456–1490, 2013.

[45] Joel A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[46] James C. R. Whittington and Rafal Bogacz. An approximation of the error back-propagation algorithm in a predictive coding network. *Neural Computation*, 29(5):1229–1262, 2017. Frontiers link.

Appendix

Content list

- Predictive coding vs. Vector Networks (Sec. A)
- VN vs. Transformers: sparse operator inference vs. state routing (Sec. B)
- Fast inference of weight atom coefficients in deep Vector Networks (Sec. C)
- Slow local dictionary updates: the Atomic-Hebb rule (Sec. D)
- Theoretical foundations and proofs (Sec. E)
- Spatial decoding: extended results (Sec. F)
- Function composition: extended results (Sec. G)
- N-body dynamics: extended results (Sec. H)
- MNIST composition: extended results (Sec. I)
- Implementation details (Sec. J)
- Hardware and storage requirements (Sec. K)

A Predictive coding vs. Vector Networks

This section clarifies the precise relation between predictive coding (PC) and Vector Networks (VN). The goal is to separate the shared optimization structure from the different architectural role played by the inferred variables.

The comparison between predictive coding (PC) and Vector Networks (VN) is easiest to understand by separating *how* inference is performed from *what role* the inferred variables play. In both cases, fast variables are inferred under fixed learned parameters by minimizing a local objective or energy [38, 16, 46, 34]. The key difference is architectural: in PC, the inferred variables are the dense layer activities themselves, whereas in VN the inferred variables reparameterize the transformation between dense layers.

In standard sparse or predictive coding, the mapping has the form

$$y = Wx. \tag{10}$$

Writing $W = [w_1, \dots, w_d]$ in terms of its columns gives

$$y = \sum_j x_j \, w_j. \tag{11}$$

Thus, changing the inferred activity x_j gates the contribution of column w_j. Inference therefore selects which input directions or features are active, but the output-side matrix W remains fixed. Put differently, PC infers the dense hidden state and only thereby determines how the fixed matrix is used; the inferred object is the layer activity itself, not an explicit parameterization of the transformation between layers.

In VN, by contrast, the effective operator is synthesized from reusable rank-1 atoms:

$$W^\star(x) = \sum_i g_i(x) \, u_i s_i^\top, \tag{12}$$

so the inter-layer transformation is first reparameterized by the sparse coefficient vector g, and inference then minimizes the layer energy with respect to that code. The coefficients g_i can be viewed as latent activities, but they are more than that: each g_i scales an entire rank-1 operator $u_i s_i^\top$, so the inferred code defines the sample-specific operator description between the two dense layers. Once $g_l^\star$ is inferred, the dense upward activity h_l (identified with x_{l+1}) is produced directly from that settled code rather than being itself inferred as an independent dense hidden state. In this sense, VN does not merely select latent features under a fixed map, but selects and composes input–output mappings, i.e. operators.

The distinction can be summarized as follows:

$$\text{PC: } y = \sum_j x_j \, w_j \qquad \text{(column / feature selection)}, \tag{13}$$

$$\text{VN: } W^\star(x) = \sum_i g_i \, u_i s_i^\top \qquad \text{(operator selection)}. \tag{14}$$

Thus both frameworks use iterative latent inference, but the inferred variables control different objects: PC infers dense states within a fixed mapping, whereas VN infers a sparse code that assembles the sample-specific operator itself and then yields the dense next-layer activity.

The same shift can be understood more concretely from a bidirectional predictive-coding (bPC) stack. For a mapping $x_l \xrightarrow{W_l} x_{l+1}$, both neighboring activities x_l and x_{l+1} are inferred in bPC. The effective use of the fixed matrix W_l is therefore jointly shaped from below by the inferred lower-layer state and from above by the inferred upper-layer state. In VN, by contrast, the dense intermediate activities are fixed-pass communication channels rather than inference variables; the model alternates dense communication layers x_l with sparse inference layers g_l, so information flows as $x_l \to g_l \to x_{l+1}$. This removes the shared-state coupling of bPC and assigns the layer-local transformation to one explicit sparse code.

Second, the fixed dense matrix W_l is replaced by a layer-local dictionary of rank-1 operators $u_{l,i}s_{l,i}^\top$, and the sample-specific transformation is assembled as

$$W_l^\star(x_l) = \sum_i g_{l,i}(x_l) \, u_{l,i} s_{l,i}^\top. \tag{15}$$

Inference therefore no longer selects only neural activities under fixed weights, but selects and composes reusable operators. Crucially, this operator selection is explicitly sparse: the coefficients g_l are inferred under an ℓ_1 penalty, and at the top layer may additionally be subject to a hard top-k budget. Only a small competitive subset of atoms therefore remains active, and a single inferred coefficient vector g_l defines the operator description between the two dense layers, rather than that transformation emerging indirectly from the interaction of two inferred neighboring states. Third, the learning rule becomes weight atom local (not neuron local):

$$\Delta S_l \propto r_l^x g_l^\top, \qquad \Delta U_l \propto r_l^h g_l^\top. \tag{16}$$

In bPC, local plasticity typically takes a pre×error or pre×post×error form, and the inferred activities are usually not constrained to select a sparse competitive subset of reusable operators, so updates remain distributed across adjacent weight matrices and are neuron- or pathway-local rather than weight-atom local. This distributed coupling leads to stronger weight entanglement across layers, because weight responsibility remains spread over neighboring inferred states and shared dense matrices. In VN, the inferred coefficients g_l update only the atoms of their own layer, so credit assignment becomes weight-atom local to the transformation itself and more directly limits such cross-layer entanglement. In this sense, the key conceptual shift is

$$\text{bPC: inference on states} \qquad \longrightarrow \qquad \text{VN: inference on transformations / operators.} \tag{17}$$

Bidirectional predictive coding therefore remains a state-inference framework: bottom-up and top-down activities are inferred under fixed forward and backward weights, and plasticity stays neuron- or pathway-local through pre×error or pre×post×error interactions. VN keeps the iterative inference structure but shifts both computation and learning onto sparse reusable operator atoms: the inferred coefficient g_l selects a competitive subset of rank-1 transformations and the same coefficient gates their weight-atom-local update. This is the sense in which VN turns compositionality into explicit responsibility over reusable weight structure.

B VN vs. Transformers: sparse operator inference vs. state routing

This section compares VN to transformer attention at the level of the adaptive variables each model uses during inference. The central distinction is between feedforward state routing in transformers and sparse operator inference in VN.

The comparison to transformers is useful because both architectures are input-adaptive, but the adaptive variables act on different objects. In a transformer, attention weights route information among token states:

$$\boldsymbol{h}_i^{\mathrm{attn}} = \sum_j \alpha_{ij}\, \boldsymbol{V}_j, \qquad \alpha_{ij} = \mathrm{softmax}\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}}\right). \tag{18}$$

The learned projection matrices W_Q, W_K, W_V, W_O and the MLP weights are fixed for a given model during inference; the input-dependent variables α_{ij} determine which value states are read and mixed. Thus, standard attention provides a powerful form of *adaptive state routing*. Unlike VN, these adaptive coefficients are generated feedforward rather than inferred by energy minimization, and they are not subject to an explicit sparsity pressure or competitive selection among reusable operator atoms.

Adaptive operator composition in VN. In VN, the input-dependent variables are sparse weight-atom coefficients. After the fast settle, the coefficients $g_l^\star(x_l)$ select rank-1 atoms and compose a sample-specific operator,

$$\boldsymbol{W}_l^\star(x_l) = \sum_k g_{l,k}^\star(x_l)\, u_{l,k} s_{l,k}^\top. \tag{19}$$

The adaptive variable therefore specifies *which reusable operators are composed by the inferred code*, not only which states are read. The same inferred coefficients later gate the slow local updates on those active atoms, so computation and plasticity are organized around the same atom-level responsibility pattern.

Low rank plays different roles. *VN.* Each weight atom is rank-1 in feature space, so the induced operator lies in

$$\boldsymbol{W}_l^\star(x_l) \in \mathrm{span}\{u_{l,k} s_{l,k}^\top\}_{k=1}^{K_l}. \tag{20}$$

Low rank constrains the operator atoms exposed after inference; sparse gains $g_{l,k}^\star(x_l)$ compose these atoms into a sample-specific operator description.

Transformer. For one attention head, pre-softmax scores form a bilinear token-token matrix,

$$\boldsymbol{S} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}} \sum_{m=1}^{d_k} \boldsymbol{Q}_{(:,m)}\, \boldsymbol{K}_{(:,m)}^\top. \tag{21}$$

This low-rank structure shapes routing capacity over tokens. Multi-head attention combines several such routing patterns, but the learned projection matrices themselves are not selected as a sparse superposition of reusable weight atoms.

Comparison summary.

Aspect	VN (operators inferred from sparse atoms)	Transformer (operators generated by attention)
Adaptive variable	Sparse coefficients $g_l^\star(x_l)$	Attention weights α_{ij}
What it acts on	Reusable rank-1 operator atoms $u_{l,k} s_{l,k}^\top$	Token/value states under fixed feature projections
How it is obtained	Iterative sparse inference under an explicit energy and sparsity penalty	Feedforward softmax attention
Does it gate learning?	Yes: the same coefficients later gate slow local updates on active atoms	No: attention weights route computation only

Competition during inference. Transformer attention weights are generated feedforward and normalized by softmax, so they redistribute mass over available states but do not arise from an explicit sparse energy minimization over reusable operator atoms. In VN, by contrast, the coefficients $g_l^\star$ are obtained by minimizing a sparsity-regularized energy, so weight atoms compete directly for responsibility under an explicit code budget.

Conceptual summary. The core distinction is that transformers generate input-dependent state-mixing operators feedforward, whereas VN infers a sparse, competitive composition of reusable rank-1 operator atoms. In VN, the converged coefficients $g_l^\star(x_l)$ select one sample-specific operator from the span

$$\mathcal{W}_l = \Big\{ \sum_k g_k \, u_{l,k} s_{l,k}^\top \;\Big|\; g \in \mathbb{R}^{K_l} \Big\}. \tag{22}$$

The same coefficients then gate the corresponding slow local updates. Transformer attention, by contrast, adapts computation by redistributing activity over states under fixed feature-space projections, without a matching responsibility-gated operator update. Low-rank structure therefore plays different roles: it governs operator composition and responsibility-gated plasticity in VN, and state routing in transformers.

C Fast inference of weight atom coefficients in deep Vector Networks

This section explains how a deep VN settles through hierarchical recurrent sparse inference. The key point is that layers are not fully optimized one at a time; instead, sparse coefficient updates are interleaved across the hierarchy through repeated upward and downward sweeps until the full stack reaches a consistent fixed point.

For layer l, the energy is

$$E_l(g_l \mid x_l, h_{l,\text{target}}) = \tfrac{1}{2}\|x_l - S_l g_l\|_2^2 + \lambda_l \|g_l\|_1 + \tfrac{1}{2}\|h_{l,\text{target}} - U_l g_l\|_2^2. \tag{23}$$

The smooth part $f_l(z) = \frac{1}{2}\|x_l - S_l z\|^2 + \frac{1}{2}\|h_{l,\text{target}} - U_l z\|^2$ has gradient

$$\nabla f_l(z) = S_l^\top (S_l z - x_l) + U_l^\top (U_l z - h_{l,\text{target}}), \tag{24}$$

where the U term is omitted when $h_{l,\text{target}}$ is not defined (top layer). A proximal-gradient update is then

$$z \leftarrow \text{prox}_{\eta\lambda_l\|\cdot\|_1}(z - \eta\,\nabla f_l(z)) = \text{soft}(z - \eta\,\nabla f_l(z),\ \eta\lambda_l)\,. \tag{25}$$

Choosing $\eta \le 1/L_f$ with $L_f = \|\, S_l^\top S_l + U_l^\top U_l \,\|_2$ yields a descent step for the coupled update. In practice we use $\eta_l \approx 1/L_f$; during the first sweep, before top-down targets have been refreshed, we optionally use a bottom-up-only step size $\eta_{s,l} \approx 1/\|S_l^\top S_l\|_2$. When a fixed code budget is required, the proximal update may be followed by a hard Top-k projection, as in Eq. (6). Thus each layer update is an ISTA/FISTA-style sparse inference step on g_l, not a full layerwise convergence before the next layer is touched.

The settle therefore proceeds as a distributed hierarchical recurrent process. Starting at the bottom layer, one sparse update step is taken for g_1; the resulting dense upward message $h_1 = \phi_1(U_1 g_1)$, identified with the next-layer input x_2, is then propagated upward so that layer 2 can take one sparse update step, and so on until the top layer is reached. The process then reverses direction: top-down targets are refreshed, $h_{l,\text{target}} \leftarrow S_{l+1} g_{l+1}$, and the same kind of sparse update is applied during a downward sweep. One complete upward and downward traversal constitutes a single inference sweep, and the network settles through many such recurrent sweeps rather than by solving each layer in isolation. Operationally, this resembles a recurrent encoder–decoder settle: activity first propagates upward toward progressively sparser codes, then reconstructed top-down targets are sent back down to refine the lower-layer coefficients. The analogy is only heuristic, however, because both passes are part of one coupled sparse inference dynamics rather than separate feedforward encoder and decoder networks.

Different schedules for this recurrent inference are possible. The simplest schedule, and the one closest to the implementation described in the main text, updates each layer once per upward pass and once per downward pass. More asynchronous variants are also possible: once a lower layer has been initialized, it can continue receiving recurrent ISTA/FISTA updates while higher layers are still being initialized, so that by the time the top layer is reached, lower layers may already have undergone several refinement steps. More generally, after an initial feedforward initialization, one can transition into a fully recurrent regime in which all layers are updated continuously at every inference time step. In all cases, the best way to understand the VN settle is as a coupled hierarchical recurrent inference dynamics in which sparse operator coefficients across all layers gradually co-adapt until the hierarchy reaches a consistent fixed point.

Other first-order variants (including accelerated proximal-gradient updates) are equally applicable as heuristic solvers with energy tracking and optional accept/reject safeguards [6]. The ℓ_1 term provides the main sparsity pressure that encourages weight-atom selection. In our experiments, hidden-layer sparsity is controlled by the soft-thresholding update itself, while any strict hard support cap is reserved for the top layer. We usually initialize $g_l = 0$ and $h_{l,\text{target}} = 0$, but the implementation also supports warm starts, resumed code states, and optional nonzero random initialization.

D Slow local dictionary updates: the Atomic-Hebb rule

This section derives the Atomic-Hebb local dictionary update, i.e. the residual×atom-coefficient rule used after the fast settle. Here $g_l^\star$ denotes the converged sparse code after fast inference has settled. The key point is that this plasticity is weight-atom local: each reusable operator atom is updated only through its own inferred coefficient and the residual available at that layer. With $g_l^\star$ fixed after the fast settle, the reconstruction term has gradient

$$\nabla_{S_l} \tfrac{1}{2}\|x_l - S_l g_l^\star\|_2^2 = -(x_l - S_l g_l^\star)\,(g_l^\star)^\top = -\,r_l^x (g_l^\star)^\top, \tag{26}$$

and the top-down term has gradient (when defined)

$$\nabla_{U_l} \tfrac{1}{2}\|h_{l,\text{target}} - U_l g_l^\star\|_2^2 = -(h_{l,\text{target}} - U_l g_l^\star)\,(g_l^\star)^\top = -\,r_l^h (g_l^\star)^\top. \tag{27}$$

Taking a direct gradient step on (S_l, U_l) yields the local residual×atom-coefficient updates used in Sec. 3.4. The subtle but important point is that Atomic-Hebb is not fully independent per atom in the strict sense, because the residuals

$$r_l^x = x_l - \sum_j g_{l,j}^\star s_{l,j}, \qquad r_l^h = h_{l,\text{target}} - \sum_j g_{l,j}^\star u_{l,j} \tag{28}$$

depend on the full sparse reconstruction and therefore on all active atoms. The update is therefore not

$$\Delta A^{(l,i)} \sim \text{local information of atom } i \tag{29}$$

in isolation. The key point instead is that credit assignment is localized at the level of *operator responsibility*. The coefficient $g_{l,i}^\star$ determines which atom participated and how strongly it contributed, while the residual measures how much unexplained error remains after the joint sparse explanation. Column-wise, the rule can therefore be read as

$$\Delta s_{l,i} \propto g_{l,i}^\star\, r_l^x, \qquad \Delta u_{l,i} \propto g_{l,i}^\star\, r_l^h. \tag{30}$$

Every active atom sees the same layer-local residual field, but scaled by its own sparse responsibility.

This is fundamentally different from backpropagation or predictive coding, where the update of a weight depends on correlations between activities of different layers, schematically

$$\Delta W_l \sim \delta_{l+1} x_l^\top, \tag{31}$$

where δ_{l+1} denotes the output error or upstream error signal carried by the next layer's activities. In VN, the update acts only on atoms of one layer, and each atom is weighted by its own inferred coefficient $g_{l,i}^\star$. The rule is therefore not "fully atom-independent," because the shared residual still depends on the joint sparse explanation, but credit assignment is localized at the level of operator responsibility. This is closely analogous to sparse coding itself: atoms compete to explain the input, the residual reflects the remaining unexplained signal, and only active atoms receive credit.

The important distinction is therefore not that each atom learns in total isolation, but that the interaction is within-layer, reconstruction-based, sparse, and competitive, rather than cross-layer, state-sharing, or dependent on gradient transport through multiple transformations. That is why the rule is still meaningfully layer-local and weight-atom-local even though the residual is shared. In the implementation, the same residual×atom-coefficient terms are used as local gradients for Adam, followed by per-column renormalization (and optional DC removal in pixel space).

Renormalization and DC removal. After each update, columns of S_l and U_l are renormalized to unit norm. For pixel-space dictionaries we optionally remove the DC component (the mean / zero-frequency content) of each S_l column before renormalization. These procedural constraints play the role of gain/variance control and keep sparsity statistics well-defined.

E Theoretical foundations and proofs

This section collects the main formal statements underlying the Vector Network architecture. It covers compositional reconstruction, the Atomic-Hebbian update, convergence of fast inference, capacity bounds, and the role of top-down coupling.

We provide formal derivations and discussions for five core elements of the Vector Network architecture: (1) why linear sparse reconstruction algebraically guarantees compositional generalization under superposition, (2) why the local Atomic-Hebbian updates are exact gradients of the equilibrium energy, (3) convergence guarantees of the fast inference dynamics, (4) the effective capacity bounds induced by sparse weight-atom inference, and (5) the top-down coupling viewed as predictive coding. Where relevant we also note gaps between the idealized theory and the practical implementation.

E.1 Compositional generalization via linear sparse reconstruction

The core empirical result motivating this section is that VNs generalize zero-shot to novel combinations of mechanisms, such as multi-bump superpositions or mixed-force OOD cases, whereas dense and factorized models fail. We formalize this in two parts: first, the algebraic preservation of superposition through the linear decoder; second, the conditions under which sparse inference recovers the correct composition.

Part 1: Superposition preservation. The VN reconstructs its output as a linear combination of dictionary atoms: $\hat{\mathbf{y}} = \sum_{i=1}^{K} g_i \mathbf{s}_i = \boldsymbol{S}\mathbf{g}$. This reconstruction is *linear* in the code $\mathbf{g}$, which immediately yields:

Proposition 1 (Superposition Preservation)**.** *Let $\mathbf{g}_1, \mathbf{g}_2 \in \mathbb{R}^K$ be sparse codes that produce outputs $\hat{\mathbf{y}}_1 = \boldsymbol{S}\mathbf{g}_1$ and $\hat{\mathbf{y}}_2 = \boldsymbol{S}\mathbf{g}_2$ respectively. Then the code $\mathbf{g}_1 + \mathbf{g}_2$ produces the exact superposition:*

$$\boldsymbol{S}(\mathbf{g}_1 + \mathbf{g}_2) = \boldsymbol{S}\mathbf{g}_1 + \boldsymbol{S}\mathbf{g}_2 = \hat{\mathbf{y}}_1 + \hat{\mathbf{y}}_2 \tag{32}$$

Proof. Immediate from linearity of matrix multiplication: $\boldsymbol{S}(\mathbf{g}_1 + \mathbf{g}_2) = \boldsymbol{S}\mathbf{g}_1 + \boldsymbol{S}\mathbf{g}_2$. □

The key insight is that any nonlinear decoder, including standard CNN decoders and factorized outer-product architectures, breaks this identity. For a nonlinear decoder D, the output $D(\mathbf{g}_1 + \mathbf{g}_2) \neq D(\mathbf{g}_1) + D(\mathbf{g}_2)$ in general, and compositional generalization becomes architecture-dependent rather than guaranteed.

For the outer-product factorized decoder specifically, the rank-1 output $f(\mathbf{x})\, g(\mathbf{y})$ applied to a superimposed input $(\mathbf{x}_1{+}\mathbf{x}_2, \mathbf{y}_1{+}\mathbf{y}_2)$ necessarily produces N^2 cross-terms instead of N correct terms, consistent with the multi-bump errors of the factorized decoders in Table 3.

Part 2: Sparse recovery of novel compositions. Proposition 1 guarantees that *if* the correct composite code $\mathbf{g}_1 + \mathbf{g}_2$ is found, the output is exact. The remaining question is whether inference can find this code when the composition was never seen during training.

Setup. Let the true generative process be $\mathbf{y} = \sum_{j=1}^{C} c_j \boldsymbol{W}_{\text{true}}^{(j)} \mathbf{x}$ where $\mathbf{c} \in \{0,1\}^C$ indicates active mechanisms. Assume the dictionary contains atoms $\boldsymbol{A}^{(i)} \approx \boldsymbol{W}_{\text{true}}^{(j)}$ for each mechanism j. Define the dynamic dictionary $\boldsymbol{\Phi}(\mathbf{x}) \in \mathbb{R}^{m \times K}$ with columns $\boldsymbol{\Phi}_i(\mathbf{x}) = \boldsymbol{A}^{(i)}\mathbf{x}$. Theorem 1 below analyzes recovery for this idealized output-side objective; it provides a guide to when sparse recovery succeeds rather than a literal analysis of VN's implemented layer-local energy ($\frac{1}{2}\|x - Sg\|^2 + \frac{1}{2}\|h_{\text{target}} - Ug\|^2 + \lambda\|g\|_1$, Eq. (3)).

Theorem 1 (Exact Compositional Recovery)**.** *Let $\mu(\boldsymbol{\Phi}(\mathbf{x}))$ denote the mutual coherence: $\mu = \max_{i \neq j} \frac{|\boldsymbol{\Phi}_i^\top \boldsymbol{\Phi}_j|}{\|\boldsymbol{\Phi}_i\| \|\boldsymbol{\Phi}_j\|}$. Assume (a) the dictionary atoms approximate the true mechanisms with bounded residual error $\mathbf{e}$, and (b) λ is chosen with $\lambda \gtrsim \|\boldsymbol{\Phi}^\top \mathbf{e}\|_\infty$. If the number of active OOD mechanisms $k_{OOD} = \|\mathbf{c}_{OOD}\|_0$ satisfies*

$$k_{OOD} < \frac{1}{2}\left(1 + \frac{1}{\mu(\boldsymbol{\Phi}(\mathbf{x}))}\right), \tag{33}$$

then the ℓ_1-regularized inference $\mathbf{g}^\star = \arg\min_{\mathbf{g}} \frac{1}{2}\|\mathbf{y} - \boldsymbol{\Phi}(\mathbf{x})\mathbf{g}\|_2^2 + \lambda\|\mathbf{g}\|_1$ recovers the true sparse support of $\mathbf{c}_{OOD}$.

Conditions under which assumptions (a) and (b) hold in practice are discussed in Remark 1.

Proof. The coherence bound is the Exact Recovery Condition for Basis Pursuit [14, 45]. In the noiseless equality-constrained case the ℓ_1 relaxation matches the ℓ_0 combinatorial minimum exactly; in the noisy LASSO regime, support recovery further requires the regularizer to dominate the residual, $\lambda \gtrsim \|\mathbf{\Phi}^\top \mathbf{e}\|_\infty$, which is the role of assumption (b). □

Remark 1 (Conditions and limitations)**.** *Theorem 1 requires two conditions that are not automatically satisfied:*

(i) ***Dictionary quality:*** *The atoms must faithfully represent the true mechanisms. In practice, this depends on the Atomic-Hebbian learning dynamics (Section E.2) and sufficient training data covering each mechanism individually.*

(ii) ***Coherence:*** *For the rank-1 atoms $A^{(i)} = u_i s_i^\top$, the columns of the dynamic dictionary factorize as $\mathbf{\Phi}_i(\mathbf{x}) = (s_i^\top \mathbf{x})\, u_i$, so the scalar projections cancel during normalization and $\mu(\mathbf{\Phi}(\mathbf{x})) = \mu(U)$ is* independent of the test input *(modulo signs of the projections). The coherence condition therefore reduces to a static incoherence requirement on the output dictionary U, which holds generically when atoms encode distinct mechanisms with different spatial/spectral signatures.*

The practical implication is that compositional generalization is an emergent *consequence of successful dictionary learning and sufficient mechanism diversity, not an unconditional architectural guarantee.*

E.2 Atomic-Hebbian updates as exact implicit gradients

The VN updates its dictionary $\boldsymbol{S}$ using a local rule $\Delta \boldsymbol{S} \propto \mathbf{r}^x (\mathbf{g}^\star)^\top$ (residual $\times$ activations) without backpropagating through the iterative inference solver. The result that such residual-times-code rules are the exact implicit gradient of an equilibrium energy is a long-established envelope-theorem argument in sparse coding [35], online dictionary learning [29], and deep equilibrium models [2]; we restate it here for completeness because it is what underlies the locality of the Atomic-Hebbian update and the validity of treating fast inference as a black box during slow learning.

Setup. Let the layer energy be $E(\mathbf{g}, \boldsymbol{S}) = f(\mathbf{g}, \boldsymbol{S}) + \lambda \|\mathbf{g}\|_1$ with smooth component $f(\mathbf{g}, \boldsymbol{S}) = \frac{1}{2}\|\mathbf{x} - \boldsymbol{S}\mathbf{g}\|_2^2$. Inference finds $\mathbf{g}^\star(\boldsymbol{S}) = \arg\min_{\mathbf{g}} E(\mathbf{g}, \boldsymbol{S})$, and learning minimizes $\mathcal{L}(\boldsymbol{S}) = E(\mathbf{g}^\star(\boldsymbol{S}), \boldsymbol{S})$.

Theorem 2 (Exact Implicit Gradients via Danskin's Theorem)**.** *Assume: (i) the columns of $\boldsymbol{S}$ are in general position (no $d{+}1$ columns lie on a common hyperplane), ensuring the LASSO minimizer $\mathbf{g}^\star(\boldsymbol{S})$ is unique [44]; (ii) the smooth component $f(\mathbf{g}, \boldsymbol{S})$ is jointly continuous in $(\mathbf{g}, \boldsymbol{S})$. Then:*

$$\frac{d\mathcal{L}}{d\boldsymbol{S}} = \left.\frac{\partial f(\mathbf{g}, \boldsymbol{S})}{\partial \boldsymbol{S}}\right|_{\mathbf{g}=\mathbf{g}^\star} = -\mathbf{r}^x (\mathbf{g}^\star)^\top \tag{34}$$

where $\mathbf{r}^x = \mathbf{x} - \boldsymbol{S}\mathbf{g}^\star$ is the reconstruction residual.

Proof. The objective $E(\mathbf{g}, \boldsymbol{S})$ is non-smooth in $\mathbf{g}$ because of the ℓ_1 term, so the classical chain rule does not apply directly. Instead, the optimality of $\mathbf{g}^\star(\boldsymbol{S})$ gives the subgradient condition $0 \in \partial_{\mathbf{g}} E(\mathbf{g}^\star, \boldsymbol{S})$, where $\partial_{\mathbf{g}}$ denotes the subdifferential. Under assumptions (i)–(ii), the generalized Danskin's theorem [11, 9] applies: the total derivative of $\mathcal{L}(\boldsymbol{S}) = E(\mathbf{g}^\star(\boldsymbol{S}), \boldsymbol{S})$ reduces to the partial derivative of the smooth component f at the optimum,

$$\frac{d\mathcal{L}}{d\boldsymbol{S}} = \left.\frac{\partial f}{\partial \boldsymbol{S}}\right|_{\mathbf{g}=\mathbf{g}^\star} = -(\mathbf{x} - \boldsymbol{S}\mathbf{g}^\star)(\mathbf{g}^\star)^\top = -\mathbf{r}^x (\mathbf{g}^\star)^\top. \tag{35}$$

The Atomic-Hebbian update $\Delta \boldsymbol{S} \propto \mathbf{r}^x (\mathbf{g}^\star)^\top$ is therefore exact gradient descent on the equilibrium energy, not a heuristic approximation. □

Remark 2 (Extension to U)**.** *The same argument applies to the interface dictionary $\boldsymbol{U}$ via the upper-consistency energy $\frac{1}{2}\|\mathbf{h}_{target} - \boldsymbol{U}\mathbf{g}\|^2$, yielding $\Delta \boldsymbol{U} \propto \mathbf{r}^h (\mathbf{g}^\star)^\top$. For the full VN energy combining both terms, the Atomic-Hebbian updates on $\boldsymbol{S}$ and $\boldsymbol{U}$ correspond to block coordinate descent on the equilibrium landscape.*

Remark 3 (Biological plausibility)**.** *The update depends only on quantities locally available at the neuron: the reconstruction residual* $\mathbf{r}^x$ *(error signal) and the converged activation* $\mathbf{g}^\star$ *(post-synaptic activity). No backward pass through the inference dynamics is required, making Atomic-Hebb compatible with the locality usually associated with Hebbian plasticity [32].*

Remark 4 (Why Atomic-Hebbian updates generalize while backprop memorizes)**.** *Theorem 2 proves that the Atomic-Hebbian update is the* correct *gradient of the equilibrium energy, but does not explain why the resulting dictionary generalizes OOD while backprop-trained decoders such as the CNN of Section 4.1 degrade. We offer the following intuition. The Atomic-Hebbian rule* $\Delta \boldsymbol{S} \propto \mathbf{r}^x(\mathbf{g}^\star)^\top$ *correlates residuals with activities that are themselves computed* from the same dictionary *via sparse inference. This self-consistency couples the learning signal to the dictionary's current representational structure: atoms are updated to reduce error in directions that the dictionary already spans. Baldi and Hornik [3] showed that for linear networks, such local update rules converge to global optima of the reconstruction objective without the saddle-point issues of generic gradient descent, providing a partial theoretical basis for why the dictionary is well-conditioned. More recently, Whittington and Bogacz [46] proved that inference-then-Hebbian-update in predictive coding networks can approximate backpropagation under specific structural conditions, suggesting the two learning rules share fixed points but may differ in their convergence basins and implicit regularization. In the VN, the two-phase structure, infer first, then update only the selected atoms, imposes an implicit curriculum: the dictionary can only learn what the sparse inference can already partially represent, preventing the global task-loss memorization that plagues end-to-end backpropagation. A complete characterization of this implicit regularization, and why it favors compositionally generalizable dictionaries, remains an open theoretical question.*

E.3 Convergence of fast inference

Setup. The layer-local energy is:

$$E_l(\mathbf{g}_l) = \underbrace{\frac{1}{2}\|\mathbf{x}_l - \boldsymbol{S}_l\mathbf{g}_l\|_2^2 + \frac{1}{2}\|\mathbf{h}_{l,\text{target}} - \boldsymbol{U}_l\mathbf{g}_l\|_2^2}_{f_l(\mathbf{g}_l)} + \lambda_l\|\mathbf{g}_l\|_1 \tag{36}$$

Theorem 3 (Convergence of Proximal Gradient Inference)**.** *The ISTA update* $\mathbf{g}_l^{(t+1)} = \operatorname{prox}_{\eta_l\lambda_l\|\cdot\|_1}\big(\mathbf{g}_l^{(t)} - \eta_l\nabla f_l(\mathbf{g}_l^{(t)})\big)$ *with step size* $\eta_l \le L_f^{-1}$*, where* $L_f = \|\boldsymbol{S}_l^\top\boldsymbol{S}_l + \boldsymbol{U}_l^\top\boldsymbol{U}_l\|_2$ *is the Lipschitz constant of* ∇f_l*, satisfies:*

$$E_l(\mathbf{g}_l^{(t+1)}) \le E_l(\mathbf{g}_l^{(t)}) - \frac{1}{2\eta_l}\|\mathbf{g}_l^{(t+1)} - \mathbf{g}_l^{(t)}\|_2^2, \tag{37}$$

so the energy decreases monotonically and converges to the global minimum at rate $O(1/t)$ *in the objective error. Using FISTA acceleration [6], the objective-error rate improves to* $O(1/t^2)$*, although the accelerated variant does not preserve per-iteration monotone descent.*

Proof. The smooth component f_l is convex quadratic with Hessian $\nabla^2 f_l = \boldsymbol{S}_l^\top\boldsymbol{S}_l + \boldsymbol{U}_l^\top\boldsymbol{U}_l$. Its gradient is L_f-Lipschitz with $L_f = \lambda_{\max}(\nabla^2 f_l)$. The descent lemma for proximal gradient methods [6] gives the stated inequality for $\eta_l \le 1/L_f$. Since E_l is bounded below and monotonically non-increasing, the iterates converge to a fixed point. Convexity of E_l ensures this fixed point is the global minimum. Convergence rates follow from [6, Theorems 3.1 and 4.4]. □

Remark 5 (Soft-k vs. hard-k regimes)**.** *Theorem 3 covers the soft-*k *regime (pure proximal* ℓ_1*, Eq.* (5)*), used in our experiments at hidden layers. The hard-*k *regime of Eq.* (6)*, applied at the top layer when a strict cardinality budget is required, composes the proximal step with a top-*k *projection. This introduces a non-convex constraint* $\|\mathbf{g}\|_0 \le k$*, so convergence to the* global *optimum is no longer guaranteed; the solver may converge to a stationary point of the combinatorial problem. We note three points:*

- *(i) The energy still decreases monotonically at each step (the top-*k *projection is a valid proximal operator for the indicator function of the* ℓ_0 *ball), ensuring convergence to a stationary point.*
- *(ii) For sufficiently incoherent dictionaries, the soft-*k *and hard-*k *regimes select the same support [45], so the two regimes are equivalent under well-conditioned dictionaries.*

(iii) Our multi-layer settling interleaves updates across layers (block coordinate descent), which inherits the monotone descent property layer-by-layer but lacks a global convergence rate guarantee for the joint objective. The closest theoretical reference is the multi-layer convolutional sparse coding framework of Sulam et al. [43], which provides convergence guarantees for a related hierarchical pursuit under incoherence conditions; extending those results to the VN's bidirectional ($\boldsymbol{S}/\boldsymbol{U}$) atom structure is an open problem.

E.4 Effective capacity bound

Standard dense layers deploy their entire parameter budget for every input. We show that the VN's sparse inference structurally restricts the per-sample hypothesis space.

Proposition 2 (Rank Bound for the Synthesized Operator)**.** *For any input* $\mathbf{x}$*, let* $\boldsymbol{W}^\star(\mathbf{x}) = \boldsymbol{U}\,\mathrm{diag}(\mathbf{g}^\star)\,\boldsymbol{S}^\top$ *denote the operator synthesized from the inferred sparse code. Its rank is bounded by the activation sparsity:*

$$\mathrm{rank}(\boldsymbol{W}^\star(\mathbf{x})) \le \|\mathbf{g}^\star\|_0 \le k \tag{38}$$

where k *is the top-*k *sparsity constraint. The corresponding per-sample parameter budget is bounded by* $k(d+m)$*, compared to* md *for a dense layer.*

Proof. $\boldsymbol{W}^\star = \boldsymbol{U}\,\mathrm{diag}(\mathbf{g}^\star)\,\boldsymbol{S}^\top$. By sub-multiplicativity: $\mathrm{rank}(\boldsymbol{W}^\star) \le \mathrm{rank}(\mathrm{diag}(\mathbf{g}^\star)) = \|\mathbf{g}^\star\|_0 \le k$. □

This rank bound characterizes the sample-specific operator induced by the inferred sparse code, not a literal forward multiplication $\boldsymbol{W}^\star(\mathbf{x})\mathbf{x}$.

Remark 6 (Capacity and reuse)**.** *The condition* $k \ll \frac{md}{m+d}$ *(where* d *is the input dimension and* m *the output dimension) ensures that the per-sample transformation is strictly lower-rank than what a dense layer could represent. Combined with a shared dictionary learned across all training samples, this forces the slow learning phase to discover reusable, compositionally generalizable atoms rather than sample-specific memorization patterns. From a statistical learning perspective, the Rademacher complexity of* k*-sparse linear models scales as* $\mathcal{O}(\sqrt{k\log(K/k)/n})$ *rather than* $\mathcal{O}(\sqrt{d/n})$ *for dense models [4], providing a formal basis for the improved generalization of sparse representations when* $k \ll d$*.*

Remark 7 (Failure conditions)**.** *The rank constraint is also a source of failure when violated:*

(i) ***Insufficient sparsity:*** *When the target function requires more than* k *independent components (e.g., a scene with* $k{+}1$ *superimposed objects), the VN necessarily under-fits because its effective weight matrix cannot span the required subspace.*

(ii) ***Dictionary collapse:*** *When atoms become near-duplicate during Atomic-Hebbian learning, the mutual coherence* $\mu(\boldsymbol{U})$ *(which equals* $\mu(\boldsymbol{\Phi}(\mathbf{x}))$ *of Theorem 1 for rank-1 atoms; Remark 1) increases and the recovery condition may be violated, causing inference to select incorrect atoms.*

(iii) ***Incomplete convergence:*** *The practical settle uses a finite number of sweeps. If inference terminates before the energy has converged sufficiently, the code* $\mathbf{g}$ *may not satisfy the optimality conditions required by Theorems 2 and 1.*

In practice, the sparse, competitive slow update counteracts condition (ii): only active atoms are updated, the ℓ_1 *penalty disfavors redundant atoms, and per-column renormalization (App. D) removes scale degeneracy. The energy-tracking accept/reject mechanism (Section 3.3) counteracts (iii). Neither condition is eliminated in the worst case.*

E.5 Top-down coupling

The VN's top-down information enters the lower-layer code $\mathbf{g}_l$ through the consistency term $\frac{1}{2}\|\mathbf{h}_{l,\mathrm{target}} - \boldsymbol{U}_l\mathbf{g}_l\|_2^2$ in the layer energy E_l. During the downward sweep (Sec. 3.3), the target is refreshed from the higher-layer reconstruction, $\mathbf{h}_{l,\mathrm{target}} \leftarrow \boldsymbol{S}_{l+1}\mathbf{g}_{l+1}$, and a coupled proximal step on $\mathbf{g}_l$ uses both bottom-up and top-down residuals.

Remark 8 (Top-down coupling via energy consistency)**.** *The top-down term contributes the gradient component* $\boldsymbol{U}_l^\top(\boldsymbol{U}_l\mathbf{g}_l - \mathbf{h}_{l,\mathrm{target}})$ *to the smooth part of* E_l*, so the proximal updates of Eqs.* (5)–(6) *automatically incorporate a top-down correction whenever* $\mathbf{h}_{l,\mathrm{target}}$ *has been refreshed. This integrates predictive-coding-style top-down information directly into the energy descent rather than as a separate update rule.*

Unlike classical predictive coding [38], the VN does not enforce strict weight transport between forward and backward pathways. The interface dictionary $\boldsymbol{U}_l$ *used in the top-down consistency term is learned independently from any forward Jacobian; implementations may also use untied dictionaries for the upward message and the downward consistency, in which case the top-down dictionary plays a role analogous to a learned feedback weight as in the Feedback Alignment principle [26]. We do not prove convergence of any such alignment for the VN's specific dynamics; an empirical ablation of the top-down coupling strength is reported in Appendix H.3.*

Remark 9 (Connections to related frameworks)**.** *The VN's inference-learning separation connects to several established theoretical frameworks:*

- ***Equilibrium models:*** *The fast-slow separation mirrors Deep Equilibrium Models [2], but with an explicit energy function and convergence guarantee rather than implicit differentiation.*
- ***Sparse coding:*** *The single-layer inference is equivalent to the LASSO / Basis Pursuit [10], with the multi-layer extension following the hierarchical sparse coding framework of Sulam et al. [43].*
- ***Energy-based models:*** *The overall architecture fits within the energy-based learning framework of LeCun et al. [24], where inference minimizes energy and learning minimizes the minimum energy.*
- ***Predictive coding:*** *The top-down coupling (Remark 8) connects to the predictive coding framework [38, 46], where higher layers predict lower-layer activity and residuals drive learning.*

F Spatial decoding: extended results

This section extends the spatial decoding benchmark from Fig. 2 with additional ablations on rank, depth, encoding, and holdout structure. The aim is to show more explicitly which parts of the VN design are responsible for the off-support robustness in the bump task.

Increasing architectural rank does not help. A rank-N factorized model trained on rank-1 data collapses to rank-1: extra streams become redundant copies (Table 3).

Table 3: **Rank ablation.** All trained on single bumps.

Architecture	$N{=}1$ OOD	$N{=}2$	$N{=}3$
Rank-1	$2.0{\times}10^{-5}$	$7.3{\times}10^{-3}$	$1.1{\times}10^{-2}$
Rank-2	$3.0{\times}10^{-5}$	$7.8{\times}10^{-3}$	$1.1{\times}10^{-2}$
Rank-3	$2.6{\times}10^{-5}$	$8.1{\times}10^{-3}$	$1.2{\times}10^{-2}$
Full 2D VN	$4.0{\times}10^{-3}$	$\mathbf{3.7{\times}10^{-4}}$	$\mathbf{4.4{\times}10^{-4}}$

VN architecture ablation. Table 4: depth 2 captures most benefit over depth 1; $k{=}8$ outperforms $k{=}16$ (sparser codes compose better); mix$=0.7$ improves VN-3L by 23% because the deeper hierarchy benefits from stronger top-down refinement of reconstruction codes.

Input encoding comparison. Table 5 shows all encoding$\times$model combinations. Fourier features benefit all models, with VN-1L improving $5\times$ and the SAE improving $10\times$ over bump. The β-VAE benefits most from Fourier-14 ($2.4{\times}10^{-4}$). The SAE shows dramatic encoding sensitivity (from $1.8{\times}10^{-2}$ to $4.2{\times}10^{-4}$), while VN-3L is robust across all encodings ($5.5{-}7.4{\times}10^{-5}$).

Table 4: **VN depth, sparsity, and top-down ablation.**

Config	Depth	k	Ratio	N=2	N=3
VN-1L	1	16	1.16×	2.65×10^{-4}	3.01×10^{-4}
VN-2L	2	8	1.08×	9.53×10^{-5}	9.88×10^{-5}
VN-3L	3	8	1.07×	9.93×10^{-5}	1.01×10^{-4}
VN-3L (mix = 0.7)	3	8	**1.06×**	$\mathbf{7.62\times10^{-5}}$	$\mathbf{7.76\times10^{-5}}$

Table 5: **Full encoding comparison** (N=2 OOD-1 MSE).

Encoding (dim)	SAE	β-VAE (β=1)	β-VAE (β=4)	VN-1L	VN-3L
Bump (56)	4.37×10^{-3}	4.75×10^{-4}	8.22×10^{-4}	2.93×10^{-4}	6.90×10^{-5}
One-hot (56)	4.76×10^{-3}	5.52×10^{-4}	5.95×10^{-4}	3.02×10^{-4}	7.43×10^{-5}
Fourier-14 (56)	4.24×10^{-4}	2.38×10^{-4}	4.88×10^{-4}	6.29×10^{-5}	5.49×10^{-5}
Fourier-7 (28)	3.60×10^{-3}	2.69×10^{-4}	3.36×10^{-4}	5.93×10^{-5}	5.90×10^{-5}
Scalar (2)	1.81×10^{-2}	1.42×10^{-3}	1.72×10^{-3}	1.04×10^{-3}	6.56×10^{-5}

Sharper bumps (σ=0.5). Table 6: the Full 2D VN holds at 1.08× ratio; on N=2 sharp bumps it is 7.7× more precise than ML-Fact. VN.

Table 6: σ=0.5: single-bump (left) and N=2 (right).

Model	OOD/ID	N=2 OOD-1
CNN	3.39×	3.09×10^{-3}
Fiete Fact.	1.05×	4.29×10^{-3}
ML-Fact. VN	0.97×	2.08×10^{-3}
Full 2D VN	1.08×	$\mathbf{2.71\times10^{-4}}$

Annular holdout. Ring-shaped OOD: Full 2D VN 1.04×, CNN 4.57×.

Emergent factorization. Removing hand-built components step by step (supervised $S \to$ Hebbian $\to$ no axis loss $\to$ flat 2D dict) preserves compositional generalization at every stage (1.31× $\to$ 1.56× $\to$ 1.47× $\to$ 1.19×).

Entangled input. When input=output (784-D flattened image), all models solve N=2 trivially ($\sim10^{-5}$) because composition reduces to autoencoding. The challenge requires a dimensionality mismatch (56-D $\to$ 784-D).

G Function composition: extended results

This section extends the function-composition results from Fig. 3 with full metric grids, qualitative Hard-OOD examples, and masked-inference details.

This appendix contains two blocks of supporting material: (i) the full evaluation grids behind both protocols – reconstruction (Appendix G.1) and masked prediction (Appendix G.2); (ii) qualitative-reconstruction galleries for both protocols (Appendix G.3) and algorithmic details of masked-FISTA inference (Appendix G.4).

G.1 Full reconstruction grid (clean training)

Table 7 reports the per-model, per-difficulty reconstruction MSE underlying the bars summarized in Fig. 3b. The masked-prediction counterpart appears in Table 8 and qualitative galleries for both protocols are collected in App. G.3.

Both tables also include K-SVD [1], which learns an overcomplete dictionary of *signal* atoms rather than operator atoms. Function composition is the one benchmark where it applies directly, because

input and output share one representation; for spatial decoding and n-body dynamics the inputs and outputs live in different spaces. Reconstruction is K-SVD's native problem: it attains the lowest ID and Easy-OOD error in Table 7, since cross-family sums stay in the span of its signal atoms, but it falls behind VN on Hard-OOD reconstruction and on every Hard-OOD masked-prediction column.

Table 7: **Full function-composition reconstruction grid** (clean training; per-model, per-difficulty MSE). K-SVD is a classical signal-dictionary baseline (App. G.1).

Model	ID	Easy	Hard
NeuralODE	2.16×10^{-1}	1.20	1.44
rSLDS	1.82×10^{-1}	1.12	1.18
DeepAE3	1.90×10^{-4}	3.37×10^{-1}	4.30×10^{-1}
SparseAE3	3.53×10^{-4}	2.71×10^{-1}	3.71×10^{-1}
RNN	5.61×10^{-4}	1.72×10^{-2}	5.34×10^{-2}
VN-1L	2.76×10^{-2}	4.26×10^{-2}	7.18×10^{-2}
Transformer	2.1×10^{-5}	2.7×10^{-4}	1.5×10^{-3}
Mamba-bottleneck	4.60×10^{-5}	1.69×10^{-2}	1.89×10^{-2}
VN-5L	7.80×10^{-5}	1.59×10^{-4}	4.21×10^{-3}
VN-3L	9.60×10^{-5}	1.73×10^{-4}	4.16×10^{-3}
Mamba	1.60×10^{-5}	3.05×10^{-4}	8.22×10^{-4}
K-SVD	2.9×10^{-9}	1.1×10^{-4}	7.7×10^{-3}

G.2 Full mask-prediction grid (clean training)

Table 8 reports the mean masked MSE for every (model, mask regime, difficulty) cell (n=600 test samples, clean training). All VN numbers use masked-FISTA inference with n_{outer}=5; all other models receive the masked input with zeroed positions, except K-SVD, for which masked prediction is classical sparse inpainting with the dictionary held fixed. The bars in Fig. 3c are the means of the four Hard-OOD columns.

G.3 Hard-OOD qualitative galleries

Aggregated MSE numbers in Tables 7 and 8 place the VN stacks first on masked Hard-OOD prediction and behind only Mamba and the Transformer on Hard-OOD reconstruction, but they do not reveal *how* the non-VN baselines fail. Two qualitatively different failure modes can produce similar mean MSEs (e.g. uniform shrinkage to the dataset mean vs. tracking the gross shape but losing high-frequency content), and these have different implications for what the model has actually learned. The galleries below inspect representative worst-case non-VN samples to surface per-baseline failure signatures.

Figures 6–7 show the reconstruction-protocol qualitative behavior. The single-example figure is the Hard-OOD sample on which the gap between VN and the three non-VN principled baselines (MAMBA-BOTTLENECK, DEEPAE3, RNN) is largest under clean reconstruction; the gallery extends this to four samples over the full 10-model roster. Across all four, DEEPAE3 collapses to a mean / linear interpolant, the RNN traces a low-frequency envelope, and MAMBA-BOTTLENECK follows the gross signal shape but misses the high-frequency detail, while both VN stacks track within the visible line width.

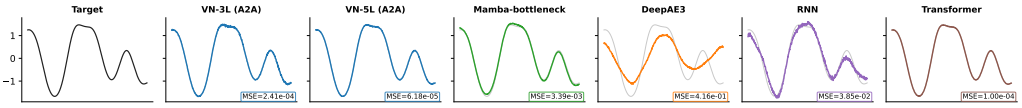


Figure 6: **Hard-OOD reconstruction (single worst non-VN sample)**. Sample selected as $\arg\max_i \rho_i$ with ρ_i=mean(MSE$_{\text{non-VN}}$)/MSE$_{\text{VN-5L}}$.

Figure 8 reports the mask-protocol counterpart of the reconstruction gallery. Three Hard-OOD samples are chosen by $\rho_i^{\text{mask}} = \text{mean}(\text{MSE}^{\text{mask}}_{\text{non-VN}})/\text{MSE}^{\text{mask}}_{\text{VN-5L}}$ on the hidden positions, with a mild ID-outlier filter. On every row both VN stacks track the signal closely through the masked tail; the autoencoder family collapses, latent- dynamics models drift off the signal entirely, and both Mamba variants catch the early observed part but diverge once the mask hides the tail.

Table 8: **Full function-composition mask-prediction grid** (mean masked MSE, clean training, n=600). Bold: best entry per (regime, difficulty) column in the Hard-OOD block. VN-3L and VN-5L take the top two places in every Hard-OOD column; on MASK_RANDOM_30/Hard the margin over the autoencoder family is more than an order of magnitude ($\sim$45$\times$). K-SVD was evaluated on Hard-OOD only (–: not evaluated).

	MASK_FORECAST_25			MASK_FORECAST_50		
Model	ID	Easy	Hard	ID	Easy	Hard
VN-3L	5.50×10^{-1}	8.06×10^{-1}	**3.91e-1**	8.84×10^{-1}	9.38×10^{-1}	**6.22e-1**
VN-5L	5.50×10^{-1}	8.04×10^{-1}	**3.92e-1**	8.85×10^{-1}	9.37×10^{-1}	**6.22e-1**
VN-1L	3.15×10^{-1}	5.90×10^{-1}	5.81×10^{-1}	8.24×10^{-1}	8.86×10^{-1}	7.73×10^{-1}
Mamba-bottleneck	5.40×10^{-1}	8.72×10^{-1}	6.24×10^{-1}	7.82×10^{-1}	9.80×10^{-1}	7.36×10^{-1}
RNN	6.88×10^{-1}	8.25×10^{-1}	1.23	8.08×10^{-1}	9.50×10^{-1}	8.21×10^{-1}
DeepAE3	1.76×10^{-1}	7.34×10^{-1}	1.49	5.22×10^{-1}	1.01	9.46×10^{-1}
SparseAE3	1.80×10^{-1}	6.73×10^{-1}	1.51	5.37×10^{-1}	9.87×10^{-1}	8.95×10^{-1}
Transformer	1.22	9.68×10^{-1}	1.48	1.07	1.00	9.57×10^{-1}
Mamba	1.34	9.80×10^{-1}	1.76	1.11	1.01	1.14
NeuralODE	1.92	1.61	1.27	2.16	1.91	1.25
rSLDS	1.92	1.56	2.34	2.11	2.17	2.15
K-SVD	–	–	4.22×10^{-1}	–	–	9.36×10^{-1}

	MASK_RANDOM_30			MASK_BLOCK_128		
Model	ID	Easy	Hard	ID	Easy	Hard
VN-3L	6.61×10^{-3}	5.71×10^{-3}	1.31×10^{-2}	4.59×10^{-1}	6.92×10^{-1}	**2.97e-1**
VN-5L	5.56×10^{-3}	4.58×10^{-3}	**1.20e-2**	4.42×10^{-1}	6.79×10^{-1}	3.04×10^{-1}
VN-1L	8.49×10^{-2}	8.36×10^{-2}	1.71×10^{-1}	2.84×10^{-1}	4.10×10^{-1}	5.12×10^{-1}
Mamba-bottleneck	1.83×10^{-2}	4.20×10^{-2}	4.35×10^{-2}	5.36×10^{-1}	8.21×10^{-1}	4.82×10^{-1}
RNN	9.65×10^{-2}	1.36×10^{-1}	2.21×10^{-1}	5.06×10^{-1}	7.82×10^{-1}	6.91×10^{-1}
DeepAE3	5.41×10^{-2}	3.85×10^{-1}	5.71×10^{-1}	2.02×10^{-1}	7.23×10^{-1}	7.90×10^{-1}
SparseAE3	5.44×10^{-2}	3.81×10^{-1}	5.25×10^{-1}	1.97×10^{-1}	6.96×10^{-1}	7.58×10^{-1}
Transformer	2.46×10^{-1}	2.36×10^{-1}	2.33×10^{-1}	8.12×10^{-1}	9.59×10^{-1}	7.62×10^{-1}
Mamba	2.35×10^{-1}	2.33×10^{-1}	2.40×10^{-1}	9.00×10^{-1}	9.95×10^{-1}	8.18×10^{-1}
NeuralODE	1.00	1.43	1.41	9.65×10^{-1}	1.60	1.92
rSLDS	6.74×10^{-1}	1.40	1.11	7.99×10^{-1}	1.66	2.12
K-SVD	–	–	9.73×10^{-2}	–	–	4.18×10^{-1}

The non-VN failures are not a single uniform phenomenon: each baseline class has its own qualitative signature (DEEPAE3 mean collapse, RNN low-frequency envelope, MAMBA-BOTTLENECK shape-without-detail; under masking, autoencoders collapse on the hidden tail and Mamba variants diverge once the observation window ends). The aggregate MSE gap therefore reflects different underlying inductive biases rather than a uniform quantitative gap. Both VN stacks consistently track the signal within the visible line width across both protocols, which matches the quantitative ranking in Tables 7–8.

G.4 Masked-FISTA VN inference

For an observed signal $x_{\text{obs}} \in \mathbb{R}^D$, mask $m \in \{0,1\}^D$, and frozen VN dictionary S, the masked imputation problem is

$$\min_z \ \tfrac{1}{2}\left\|m \odot (x_{\text{obs}} - Sz)\right\|_2^2 \ + \ \lambda \left\|z\right\|_1, \tag{39}$$

where the observation-mismatch loss is restricted to the observed positions and hidden positions are filled by the sparse linear synthesis Sz.

Masked-FISTA VN is an outer imputation loop around the ordinary VN inference call VN_INFER$(\cdot)$ that solves Eq. (39) without retraining any VN weights (Algorithm 2).

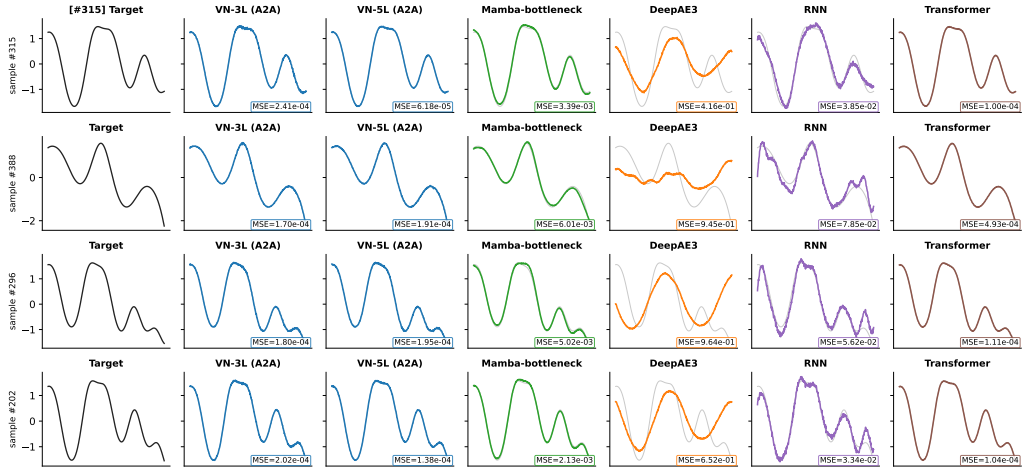


Figure 7: **Hard-OOD reconstruction gallery** (four worst-case non-VN samples).

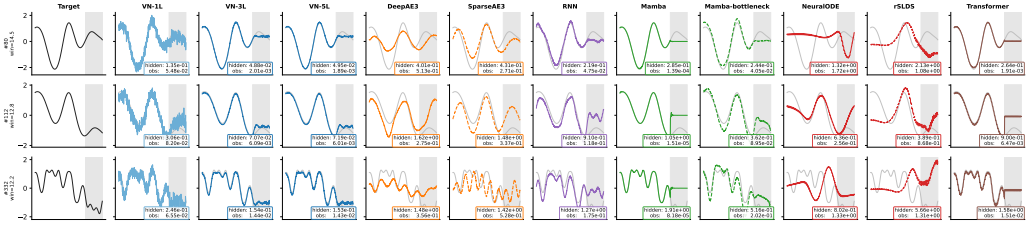


Figure 8: **Hard-OOD masked-prediction gallery** (MASK_FORECAST_25). Three worst-case non-VN samples over the full ten-comparator roster. Grey curves: target; grey shading: hidden region; colored curves: model output; per-panel numbers: MSE on hidden positions and on observed positions.

Algorithm 2 Masked-FISTA VN inference

Require: Observed signal $x_{\text{obs}} \in \mathbb{R}^D$, mask $m \in \{0,1\}^D$, frozen VN stack with dictionary S, outer iterations n_{outer}=5

$z^{(0)} \leftarrow \text{VN_INFER}(m \odot x_{\text{obs}})$
for $k = 1, \ldots, n_{\text{outer}}$ **do**
  $x^{(k)} \leftarrow m \odot x_{\text{obs}} + (1 - m) \odot S\, z^{(k-1)}$
  $z^{(k)} \leftarrow \text{VN_INFER}(x^{(k)})$
return $\hat{x} = S\, z^{(n_{\text{outer}})}$

At convergence, observed positions match x_{obs} exactly and hidden positions are filled by the sparse linear synthesis $S\, z^\star$. The fixed point coincides with the minimizer of (39) because each inner call already solves an ℓ_1-regularized residual against a fully observed signal. We fix n_{outer}=5 throughout the paper; empirically five outer steps are sufficient for the masked residual to fall below the FISTA tolerance on every VN model we tested. No VN weight is updated in this procedure, so the method transfers the reconstruction-trained dictionary to the prediction setting at a $\approx 5\times$ inference cost and *zero* retraining cost. Non-VN baselines are evaluated with hidden positions set to zero, exactly matching how their recurrent / state-space / autoencoder decoders would see masked evidence in deployment.

H N-body dynamics: extended results

This section extends the n-body results from Fig. 4 with force-law details, rollout analyses, and additional OOD probes. These diagnostics test whether the same compositional advantages seen in one-step prediction remain visible under longer-horizon physical dynamics.

This appendix consolidates the n-body details: force-law definitions and design rationale (Sec. H.1), the body-dropout Pareto analysis (Sec. H.2), the top-down coupling ablation (Sec. H.3), and trajectory rollouts (Sec. H.4).

H.1 Force law definitions

All simulations use five unit-mass particles in a 2D box of side length 5. Initial positions are drawn uniformly from $[-2.5, 2.5]^2$; initial velocities from $\mathcal{N}(0, 0.5^2)$.

The four force primitives acting on particle i due to particle j are:

$$\mathbf{F}_{\text{grav}} = -G\frac{\hat{\mathbf{r}}_{ij}}{(r_{ij}^2 + \epsilon^2)}, \qquad G = 1.0,\ \epsilon = 0.1 \tag{40}$$

$$\mathbf{F}_{\text{spring}} = -k\,(r_{ij} - r_0)\,\hat{\mathbf{r}}_{ij}, \qquad k = 0.5,\ r_0 = 1.0 \tag{41}$$

$$\mathbf{F}_{\text{drag}} = -\gamma\,\mathbf{v}_i, \qquad \gamma = 0.3 \tag{42}$$

$$\mathbf{F}_{\text{lor}} = \omega\,\big(-v_{i,y},\, v_{i,x}\big), \qquad \omega = 2.0 \tag{43}$$

where $r_{ij} = \|\mathbf{r}_i - \mathbf{r}_j\|$ and $\hat{\mathbf{r}}_{ij}$ is the unit direction from j to i. A soft boundary dampens particles that drift beyond box size (position scaled by 0.98, velocity by 0.9). The integrator is fourth-order Runge-Kutta with $\Delta t = 0.005$.

Choice of force forms. The four primitives were selected so that each carries a distinct *functional class*, ensuring that the compositional task tests recombination of qualitatively different interactions rather than sign- or scale-flipped variants of the same dynamics.

Gravity (Eq. 40). A softened Newtonian $1/r^2$ attraction between unit masses, the canonical long-range inverse-square force. The softening $\epsilon = 0.1$ removes the $r \to 0$ singularity without altering the long-range form.

Spring (Eq. 41). Hooke's law with rest length $r_0 = 1$: linear in displacement, equilibrium-seeking, qualitatively unlike any inverse-power law, and the only primitive whose sign flips at a finite separation.

Drag (Eq. 42). Linear viscous damping in the Stokes regime: a function of the focus body's own velocity rather than relative position, and the only dissipative primitive. This is the one force that breaks energy conservation, deliberately included to rule out approaches that assume conservative dynamics (e.g. Hamiltonian/Lagrangian neural networks).

Lorentz rotation (Eq. 43). A velocity-dependent rotation field of the form $\omega(-v_{i,y}, v_{i,x})$. Unlike the other three primitives, this force depends directly on the current velocity rather than relative displacement, making it qualitatively distinct from gravity, spring, and drag. Its role in the benchmark is to test whether models can recombine positional and velocity-dependent mechanisms under unseen compositions without collapsing them into one shared dense interaction.

H.2 Body-dropout Pareto on the n-body benchmark

The ID–OOD comparison in the main figure (Fig. 4c) averages over body counts. Aggregation hides whether a model that looks competitive at training-time body count $n = 5$ actually holds up when bodies are dropped to $n = 4, 3$, and whether degradation costs ID, OOD, or both. To expose this we plot, for each model, the joint trajectory through (Median ID, Median OOD) space as n shrinks; the resulting Pareto figure makes each baseline's failure mode geometrically diagnosable.

Figure 9 places each model at (Median ID MSE, Median OOD MSE) on log–log axes, with three points per model, one for each of $n = 5, 4, 3$, connected by a thin line. Large marker is $n = 5$, small marker $n = 3$. The diagonal is the no-gap line (OOD = ID); points below it are structurally impossible because the OOD pool is a strict superset of the ID set.

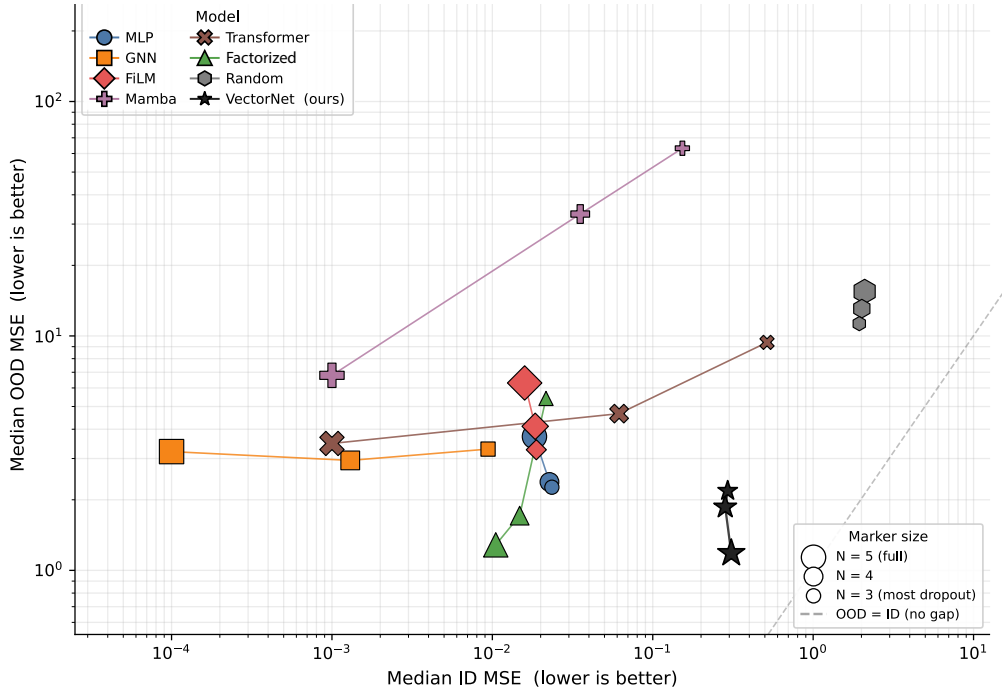


Figure 9: **Body-dropout Pareto on the n-body benchmark.** Median ID vs. Median OOD MSE for each model, evaluated at n=5, 4, 3 bodies (large $\rightarrow$ small marker).

Each model traces a characteristic geometric signature: VECTOR NETWORK (black star, lower-right) is the only model that stays in the low-OOD region as n shrinks; its trajectory barely moves. MAMBA (purple) climbs most steeply, an excellent $n = 5$ ID fit, but roughly an order of magnitude OOD growth (and two orders in ID) as bodies are dropped, the signature of an architecture that memorizes the training-time problem size. TRANSFORMER (brown) blows up *diagonally*: both ID and OOD degrade together, ID by $\sim$500$\times$ and OOD by $\sim$3$\times$. GNN (orange) moves *horizontally*, its ID error grows by $\sim$100$\times$ while OOD barely changes, the opposite kind of memorization (training body count baked into the message passing). MLP, FILM, and FACTORIZED cluster mid-plot, neither winning ID nor OOD; RANDOM (grey) fixes the predict-the-mean ceiling and is by construction insensitive to n.

Aggregated MSE hides direction: most baselines that look "competitive" at fixed n collapse along a specific axis (ID, OOD, or both) as n shifts, and each axis corresponds to a recognizable failure mode. Vector Network is the only model that tolerates body dropout while remaining in the low-OOD region, evidence that the learned compositional structure does not bake the training-time problem size into the operator the network synthesizes per sample.

H.3 Top-down coupling ablation

Recipe-development sweeps consistently identified three qualitative properties of the top-down coupling. First, a small global feedback weight ($\lambda_{\text{td}} \approx 0.3$) yields a small but systematic OOD improvement at modest ID cost. Second, splitting the top-down projection per force backfires under the per-force training budget, since each force-specific projection sees only a quarter of the data. Third, the asymmetry between top-down and bottom-up coupling is real, inverting the direction degrades every metric we tracked. We therefore use $\lambda_{\text{td}} = 0.3$ as a single global feedback weight in all reported VN N-body results.

H.4 Trajectory rollout analysis

The N-body benchmark evaluates single-step acceleration prediction, which leaves open whether good single-step predictions actually *compose* into physically plausible long-horizon dynamics.

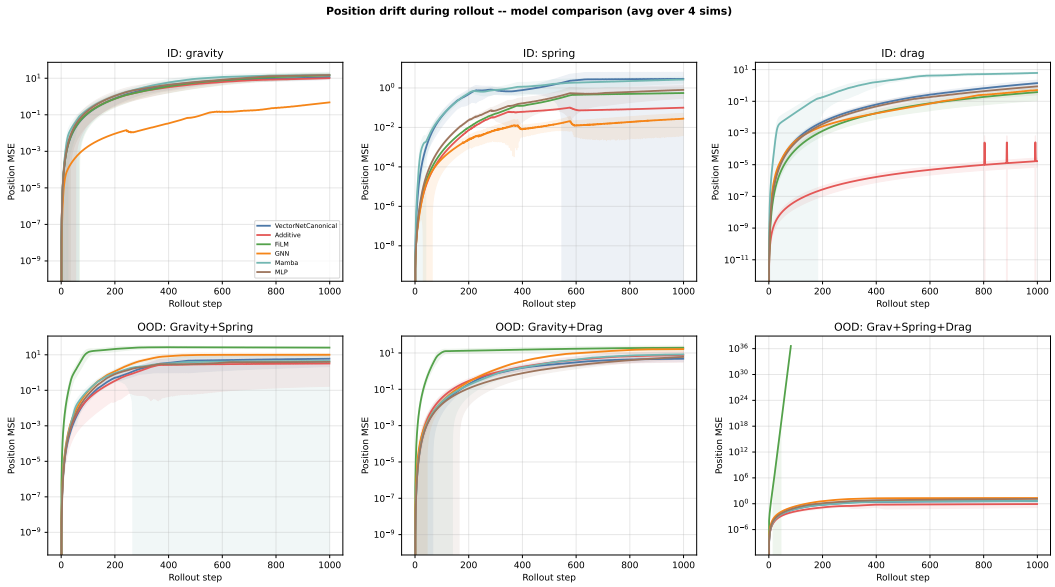


Figure 10: **Position drift during rollout** (5 s, Runge–Kutta, avg. over 4 initial conditions, shading $= \pm 1\sigma$). Top row: ID single forces; bottom row: OOD compositions. Vector Network (blue) has the lowest average drift on five of six panels.

Compositional generalization in this setting is only meaningful if the per-step predictions add up to stable trajectories, so we additionally roll each model forward in time and measure how the error accumulates.

Starting from the true state at $t = 0$, we iterate (i) construct the 22-dim per-body input features from the current predicted state, (ii) query the model for the instantaneous acceleration $\hat{\mathbf{a}}_i$ of each body, (iii) advance velocities and positions with fourth-order Runge-Kutta with $\Delta t = 0.005$ described in App. H.1, and (iv) apply the same soft-boundary damping as the simulator (position scaled by 0.98, velocity by 0.9 whenever a body drifts outside the box). This is an *open-loop* rollout: the model sees its own predicted state at every step, so errors compound. The rollout covers 999 steps (5 s of simulated time, $5\times$ longer than a training sample).

Figure 10 shows position MSE vs. rollout step on the ID single-force conditions (top row) and the OOD compositions (bottom row), with shading showing $\pm 1\sigma$ over four initial conditions. Vector Network (blue) maintains the lowest drift on five of six panels, consistent with its strongest single-step accuracy on the same conditions. The Factorized baseline is the most informative counter-example: despite very low single-step MSE, it accumulates position error faster than VN after ~ 200 steps, because its per-force sub-networks memorize local dynamics but cannot self-correct once the accumulated state drifts outside the training distribution. The remaining baselines spread across the panels without a single model approaching VN's drift envelope.

Figure 11 visualizes ground-truth (top row) vs. model-predicted body trajectories (subsequent rows) for three OOD compositions. For each condition we select one simulation to display from 20 candidates using three filters: (i) all bodies of the ground-truth trajectory stay within $0.95 \cdot L_{\text{box}}$ of the origin (no boundary damping); (ii) bodies traverse an average of at least 2 box units over the rollout (nontrivial dynamics); and (iii) among candidates that pass both, we pick the one that minimizes Vector Network's rollout MSE, a per-condition best case for our model that nonetheless uses the same initial state for every baseline. The figures show that VN trajectories largely overlap ground truth, while baseline trajectories diverge in characteristically different ways (identity-style straight lines, boundary zig-zag, FiLM divergence).

Single-step accuracy translates into long-horizon stability for Vector Network, as shown by the lowest drift on five of six drift panels and the closest qualitative match to ground truth on every OOD trajectory. The visible rollout artifacts (straight-line segments, soft-boundary zig-zag, FiLM

divergence) reflect open-loop integration of single-step predictors and do not alter the relative ordering of models in Fig. 10; the trajectory figures are therefore best read as a qualitative complement to the drift curves rather than as a standalone evaluation. Crucially, the single-step *vs.* rollout comparison shows that low single-step MSE alone (e.g. Factorized) is not sufficient for compositional stability over long horizons.

I MNIST composition: extended results

This section extends the MNIST results from Fig. 5 with representation, sparsity, and convergence diagnostics. The goal is to show how the learned sparse codes use representational budget and how stable the inferred supports become during fast settling.

Figure 12 provides additional diagnostics for the MNIST experiments beyond the main-paper OOD reconstruction errors. These panels quantify how the learned sparse codes use representational budget, how quickly the inferred supports stabilize during fast settling, and how sparsity changes across depth. Together they show that the MNIST VNs do not merely achieve low OOD error, but do so with progressively more selective deeper codes whose active subsets lock in reliably during inference.

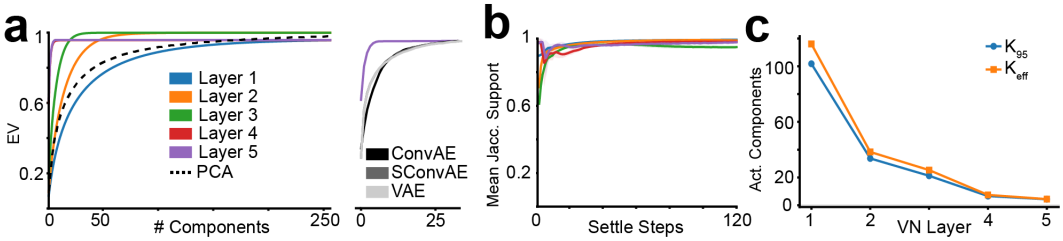


Figure 12: **Extended MNIST diagnostics. (a)** Explained-variance versus representational-budget curves. The left panel compares VN layers against PCA by plotting the fraction of variance explained as the number of active atoms or components increases, showing how deeper VN layers trade denser reconstructions for progressively more selective sparse codes. The right panel compares the top-layer VN against autoencoder baselines at matched representational budgets, making explicit that the VN reaches strong reconstruction quality with fewer active components than dense latent baselines. **(b)** Support stability across 120 fast-settle sweeps, measured by overlap between successive sparse supports. This panel shows how quickly the active atom set locks in during recurrent inference: early sweeps allow support reconfiguration, after which the inferred sparse code becomes stable while the residual continues to refine. **(c)** Layerwise sparsity statistics, reported as k_{95} and k_{eff}. Here k_{95} is the number of active atoms needed to recover 95% of the final explained variance, while k_{eff} summarizes the effective support size of the inferred code. The trend across layers shows the intended compression cascade: higher layers operate with tighter active subsets and more selective atom usage than lower layers.

J Implementation details

This section collects the VN architecture, training hyperparameters, and baseline models used in each benchmark. VN widths, depths, and atom counts as well as the baseline roster differ across benchmarks, so each benchmark is treated separately. Every subsection follows the same structure: *Inputs and training*, *VN configuration*, and *Baselines*.

J.1 Spatial composition (Sec. 4.1)

Inputs and training. All models map a 56-D bump coordinate encoding (two stacked 1-D Gaussian markers, following [25]) to a 28×28 output grid. We use N_{train}=3,000 single-bump samples with σ=1.0, dark-mode targets $1-\exp(-d^2/2\sigma^2)$, and a square holdout region $[10, 18]^2$. Every model is trained for 120 epochs with AdamW (η=10^{-3}, weight decay 10^{-5}), batch size 32.

VN configuration. We use two variants. **VN Dec.** retains the rank-1 outer-product readout of the factorized decoder but replaces the factorized atom learning with a two-layer hierarchical VN stack (l_1=128, l_2=64 atoms; k_1=16, k_2=8; 15 settle steps per layer) trained with Hebbian dictionary updates (η_U=2×10^{-4}), reconstruction $\hat{I} = \mathbf{a}_x\mathbf{a}_y^\top$ ($\sim$0.1 M params). **VN Enc.+Dec.** removes the factorization prior entirely: a 56→256→128 VN encoder, hard top-k=16 sparsification, a single Hebbian dictionary layer (128 atoms, η_S=10^{-3}, 5 settle steps) and reconstruction $\hat{I} = \mathbf{c}\,\mathbf{S}^\top + b$ ($\sim$0.17 M params). Encoder depth is the variable swept in Fig. 2d.

Baselines. We follow the architectures of [25] for the CNN and factorized decoders and add a variational variant.

- **CNN Dec.** [25] ($\sim$0.1–0.5 M params). Depth-d ConvTranspose2d stack (hidden 64, ReLU) mapping $56 \to 7\times7 \to 14\times14 \to 28\times28$. We use the depth-8 variant in Fig. 2b,c and $d \in \{2, 3, 4, 5, 6, 8\}$ in Fig. 2d.
- **Fact. Dec.** [25] ($\sim$0.6 M params). Per-axis encoder banks (x, y atoms $= 64$ each, top-k_x=k_y=6, hidden 256, 3 encoder layers) with a rank-1 outer-product readout $\hat{I} = \mathbf{a}_x\mathbf{a}_y^\top$, trained with the include-stripes augmentation of their Section 4.2.
- **VAE Dec.** ($\sim$0.13 M params). Encoder $56 \to 256 \to 256 \to (\mu, \log\sigma) \to 64$-D latent followed by a 6-stage ConvTranspose decoder (hidden 64); KL weight β=1.

J.2 Function composition (Sec. 4.2)

Inputs and training. Inputs are length-512 signals over $t\in[0, 2\pi]$. All models are trained on 300,000 ID training signals (batch 512 on a single GPU, MSE loss against the clean target) and evaluated on n=6,000 samples per OOD level. The same clean-trained checkpoint is reused under both the reconstruction protocol (Easy/Hard OOD) and the masked-prediction protocol; for VN the masked-prediction inference is the iterative masked-FISTA loop of App. G.4. Sequence baselines reshape the 512-D input as 64 tokens of 8 features.

VN configuration. All three variants share the layer-local fast-then-slow loop with a per-layer Top-K sparsity gate. VN-1L is the single-layer VN; VN-3L and VN-5L stack the FISTA solver with energy-based restart, an identity-initialized square mixing matrix U, and joint-from-start training (no staged layer activation). Settle horizons are $T_{\max}$=5 during training and $T_{\max}\in[1, 5]$ at evaluation. Optimizer: AdamW, weight decay 10^{-6}, 5,000 steps for the stacks (4,000 for VN-1L). The per-layer group-sparsity strength λ is $\{0.005,\ 0.01,\ 0.005,\ 0.005,\ 0.003\}$ for the five layers of VN-5L (the first three values match VN-3L). Sizes, sparsities and learning rates are summarized in Table 9.

Table 9: **VN configurations for function composition.** L is the layer count, K the dictionary size, k_{top} the per-sample Top-K active set at initialization before FISTA steps, and η_S the AdamW learning rate on S and U.

Model	L	K per layer	k_{top} per layer	η_S per layer ($\times10^{-4}$)	Trainable params
VN-1L	1	1024	128	10	0.52M
VN-3L	3	[256, 512, 1024]	[410, 256, 64]	[5, 7, 10]	2.88M
VN-5L	5	[256, 512, 1024, 1024, 1024]	[410, 256, 64, 32, 8]	[5, 7, 10, 12, 15]	5.71M

Baselines. All baselines are autoencoder-style (forward pass returns a length-512 reconstruction), trained with AdamW, batch 512, gradient-norm clipped at 1.0, MSE loss against the clean target. AE / Mamba / NeuralODE / rSLDS are run for 4,000 steps at learning rate 10^{-3}; the Transformer at 5×10^{-4}, weight decay 10^{-5} (more stable on the masked-input shift); the RNN at 10^{-3}. None of the baselines carry a sparsity prior except SPARSEAE3 (ℓ_1 penalty λ=10^{-4} on the bottleneck code). Architectures and trainable parameter counts are listed in Table 10; sequence baselines tokenize the input as 64×8 patches. K-SVD [1] is the exception to this recipe: a classical dictionary-learning baseline fit with Batch-OMP sparse coding and approximate K-SVD updates [40] rather than gradient training, which synthesizes the length-512 output directly from its dictionary.

Table 10: **Function-composition baselines: architectures and trainable parameter counts.** All baselines decode back to length 512 via a per-token output projection (or full-signal MLP for the AE family; dictionary synthesis for K-SVD).

Model	Architecture	Params
DEEPAE3	MLP enc/dec $512 \to 1024 \to 1024 \to 512 \to 256$ + mirror, ReLU	4.46M
SPARSEAE3	DEEPAE3 architecture, ℓ_1 penalty $\lambda = 10^{-4}$ on z	4.46M
RNN	4-layer bidirectional LSTM enc/dec ($d_h = 256$); pooled 256-d latent; 32-step decode to 16×32	4.41M
NEURALODE	GRU encoder ($d_{\text{lat}} = 128$), $128 \to 256 \to 248$ ODE field, RK4 over 64 steps + MLP dec	0.36M
RSLDS	GRU enc, soft 6-mode gate $\to$ per-mode affine $\dot{z} = A_k z + b_k$ + MLP dec	0.16M
MAMBA	4 selective-scan blocks ($d_m = 256$, $d_{\text{state}} = 32$, expand 2, $d_{\text{conv}} = 4$) over 64 tokens	2.72M
MAMBA-BOTTLENECK	MAMBA + global LayerNorm $\to$ Linear $8192 \to 256 \to 8192$ bottleneck	4.95M
TRANSFORMER	8 pre-norm encoder layers ($d_m = 256$, 8 heads, $d_{\text{ff}} = 256$, GELU) over 64 tokens, learned positional embedding	2.54M
K-SVD	Signal dictionary of 2048 atoms, Batch-OMP coding with 64 nonzeros, approximate K-SVD updates, atom replacement, overcomplete-DCT initialization	1.05M

J.3 N-body force composition (Sec. 4.3)

Inputs and training. All models share the same per-body 22-D feature: relative position Δp and relative velocity Δv to each of the four neighbors (16 d), the body's own velocity (2 d), and a 4-d one-hot force-context vector indicating which of $\{\text{GRAVITY}, \text{SPRING}, \text{DRAG}, \text{LORENTZ}\}$ are active in the simulation. The output is the 2-d acceleration. Targets are normalized per-force to unit standard deviation; the loss is per-force-balanced Huber. Every model is trained for 80 epochs with AdamW ($\eta = 10^{-3}$, weight decay 10^{-5}), batch size 512, cosine LR schedule and early stopping. We use 500 simulations per force per body count for training; OOD compositions are held out entirely. Input dimensions are $d_{\text{in}} = 22$, $d_{\text{out}} = 2$.

VN configuration. Three VN layers with $S \in \mathbb{R}^{d_{\text{in}} \times K}$ analysis dictionaries and learned $U \in \mathbb{R}^{K \times K}$ code banks; widths $(K_1, K_2, K_3) = (128, 256, 256)$, top-$k$ activations $(k_1, k_2, k_3) = (32, 16, 8)$, group-sparsity coefficients $\lambda_1 = 5 \times 10^{-4}$, $\lambda_2 = 2.5 \times 10^{-4}$, $\lambda_3 = 10^{-4}$. Each layer runs 5 fast-settle steps with adaptive-k and sigmoid gain normalization. Slow updates use η_S growing with depth (5×10^{-4}, 7×10^{-4}, 10^{-3}) and $\eta_U = 10^{-4}$. Top-down feedback is a single zero-initialized linear correction with weight 0.3; layers 2 and 3 are trained jointly. Each layer's code $c = g \cdot (yU)$ is projected to a 128-d feature, the three features are concatenated, and a 4-way force-conditioned head (LN $\to$ 2, GELU) emits the per-force acceleration ($\sim$262k params).

Baselines. All baselines share the input/output contract and the optimizer/schedule described above; only the body of the model differs. Architectures and trainable parameter counts are listed in Table 11. The FACTORIZED baseline is by construction matched to the simulator's compositional structure (per-force MLP heads summed over active forces) and is the strongest non-VN baseline; RANDOM predicts $\hat{a} = 0$ and fixes the predict-the-mean ceiling.

Table 11: **N-body baselines: architectures and trainable parameter counts.** All baselines share the 22-d $\to$ 2-d input/output contract and the AdamW/cosine training recipe of App. J.3.

Model	Architecture	Params
MLP	4-layer pre-norm MLP, hidden 256, GELU	$\sim$204k
GNN	6-layer message-passing network, node and edge dim 64 each, sum-aggregation; 4×2-d $(\Delta p, \Delta v)$ pairs unpacked into edge messages	$\sim$131k
FILM	4-layer MLP (hidden 256) FiLM-modulated by an embedding of the 4-d force context	$\sim$279k
MAMBA	4-layer selective-scan SSM-Mamba block ($d_m = 128$, $d_{\text{state}} = 32$, expand $\times 2$), input-dependent gating + causal depthwise conv	$\sim$189k
TRANSFORMER	4-layer pre-norm encoder ($d_m = 64$, 4 heads, FFN 256, GELU, dropout 0), learned positional embeddings	$\sim$211k
FACTORIZED	Per-force MLP head (hidden 128, 3 layers, GELU) with outputs summed over active forces	$\sim$143k
RANDOM	Predicts $\hat{a} = 0$ (matched-mean baseline)	—

J.4 MNIST composition (Sec. 4.4)

Inputs and training. We use two MNIST-derived benchmarks. For the static digit-composition benchmark, every sample is a standard 28×28 MNIST image stored as a dense 784-D vector and trained from the standard derived MNIST train split. Training uses Adam with batch size 256 for $50{,}000$ steps, weight decay 10^{-6}, $\tau_E{=}5\times10^{-4}$, FISTA inference with $T_{\max}{=}120$, energy-based restart, and the same bad-energy safeguards used in the other deep VN runs. For the RTS motion benchmark, the target is the transformation-induced difference image $dx = x_t - x_{t-1}$ on a 40×40 canvas. The ID `dx` train/test set uses $N_{\text{train}}{=}60{,}000$, $N_{\text{test}}{=}10{,}000$, $T{=}21$, translation range ±5 pixels at unit step, rotation range $\pm15^\circ$, and in/out scaling $(1.5, 0.55)$; OOD motion sets are generated separately with the same rendering conventions. The reported RTS motion evaluation uses the paper checkpoint exported at step $40{,}000$, with batch size 256 and 512 examples per OOD set.

VN configuration. The best static-content model is a 5-layer VN with $K = (4096, 1024, 1024, 1024, 1024)$, $m = (784, 640, 320, 240, 240)$, $k_{\text{top}} = (4096, 1024, 1024, 1024, 7)$, and $\lambda = (0.07, 0.12, 0.12, 0.24, 0.0025)$. All layers use identity ϕ_l; the sparsity penalties are $(\ell_1, \text{Cauchy}, \text{Cauchy}, \text{Cauchy}, \ell_1)$ with Cauchy scales $(0.05, 0.03, 0.08, 0.10, 0.05)$. The content run uses Adam slow updates with $\eta_S{=}4\times10^{-4}$, $\eta_U{=}10^{-5}$, and $\eta_V{=}4\times10^{-4}$. The RTS motion checkpoint uses the same 5-layer stack structure and the same per-layer sparsity / inference settings when reconstructing dx; at evaluation, the top-down (TD) variant uses the full 5-layer recurrent stack, while the 1-layer ablation disables TD and keeps only the first layer active.

Baselines.

- **Static digit benchmark.** We compare against three convolutional autoencoder families trained on the same derived MNIST inputs and matched either by ID performance or by bottleneck sparsity. CONVAE uses a $28\times28 \to 14\times14 \to 7\times7$ convolutional encoder with channel widths $(32, 64, 128, 128)$, a fully connected bottleneck, and a mirrored transposed-convolution decoder. SPARSECONVAE uses the same encoder/decoder trunk but applies a hard TopK mask to the bottleneck code. VAE reuses the same 7×7 convolutional trunk and decodes from a diagonal-Gaussian latent parameterized by $(\mu, \log\sigma^2)$, with an optional hard TopK mask on μ in the sparsity-matched setting. These baselines use Adam, batch size 256, learning rate 10^{-3}, and early stopping on test explained variance.
- **RTS motion benchmark.** Motion baselines use the corresponding 40×40 `dx` models: CONVAE5X5BOTTLENECK, SPARSECONVAE5X5BOTTLENECK, and CONVVAE5X5. Here the encoder downsamples $40\times40 \to 20\times20 \to 10\times10 \to 5\times5$ with channel widths $(32, 64, 128, 128, 128)$ before a fully connected bottleneck and mirrored decoder; the sparse AE again imposes a hard TopK bottleneck, and the VAE uses the same trunk with a diagonal-Gaussian latent and optional TopK masking in the sparsity-matched setting. All three are trained on the RTS40 ID `dx` train/test split with Adam, batch size 256, learning rate 10^{-3}, and early stopping.

K Hardware and storage requirements

This section summarizes the practical compute and storage requirements for reproducing the experiments in this paper. The intent is to make clear that the reported results do not rely on distributed training or unusually large hardware budgets.

All experiments in this paper were run on consumer-grade single-GPU workstations. The original benchmarks were developed and evaluated on an NVIDIA RTX 2080 Ti (11 GB VRAM); subsequent runs were reproduced on a modern NVIDIA RTX 5090, which has substantial memory headroom and shortens wall-clock training time but does not change any reported result. No multi-GPU training, distributed compute, or cluster infrastructure is required.

System memory of 16 GB is sufficient for every experiment in the paper; the machines we used were equipped with 64 GB, but peak working-set memory never exceeded the 16 GB budget. Disk-storage requirements are negligible: roughly 1 GB of free space is enough to hold the trained-model checkpoints, intermediate activations, diagnostic JSON bundles, and rendered figures for all experiments combined.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction (Sec. 1) describe three core contributions , the rank-1 weight-atom architecture, energy-based sparse weight-atom inference, and Atomic-Hebbian local updates , and claim about-an-order-of-magnitude lower OOD error on compositional generalization. These claims are supported by four benchmarks (Sec. 4: spatial decoding, function composition, N-body forces, MNIST composition) and the formal results in App. E.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Discussion (Sec. 5) explicitly identifies the recurrent sparse settle as the main current limitation (slower than a single feedforward pass) and points to learned amortized inference as the immediate path forward.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All theorems and propositions are stated and proved in App. E (Proposition 1, Theorem 1, Theorem 2, Theorem 3, Proposition 2); assumptions are stated explicitly and remarks discuss conditions and failure modes.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The architecture, energy, inference (Eqs. (5), (6)), and learning rules are specified in Sec. 3; benchmark protocols (training/test splits, OOD definitions, baselines) are described in Sec. 4; per-benchmark VN architecture, training hyperparameters, and baseline descriptions are collected in App. J (with VN configurations in Table 9 and baseline architectures and parameter counts in Tables 10, 11); extended results, ablations, and qualitative diagnostics appear in Apps. G, F, H, and I.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Code will be released publicly after acceptance, once the research repository has been cleaned up for distribution. The data sources are either public (MNIST [23], the 2D Gaussian-bump benchmark adapted from [25]) or fully specified procedurally by this paper (function-composition synthetic signals in Sec. 4.2, N-body simulator in App. H.1), so the experimental setup is reproducible from the paper alone in the meantime.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training, test, and OOD splits and the optimiser choice (AdamW on local residual$\times$code gradients, see Sec. 3.4) are described in Sec. 4; per-benchmark VN architecture, training hyperparameters, and baseline configurations are collected in App. J, with VN sizes/sparsities/learning rates in Table 9 and baseline rosters in Tables 10, 11; depth/sparsity/encoding sweeps, force-law parameters, and additional diagnostics are reported in Apps. G, F, H, I, and H.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The aggregated main-body figures report 95% confidence intervals over multiple training seeds: spatial decoding (Fig. 2c), function composition (Fig. 3b, c), N-body force composition (Fig. 4b), and MNIST composition (Fig. 5c). The body-dropout Pareto (Fig. 9) instead reports median ID vs. median OOD MSE for each body count. Spatial-decoding aggregated results in App. F additionally report mean $\pm$ std over 3 seeds.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware, memory, and disk-storage requirements for reproducing every experiment in the paper are summarized in App. K; all experiments run on a single consumer-grade GPU (e.g., RTX 2080 Ti or RTX 5090) with 16 GB system memory and $\sim$1 GB free disk space.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: We reviewed the NeurIPS Code of Ethics and found this work compliant.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: This work does not pose any immediate significant societal impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The only external assets used are publicly released datasets and benchmarks: MNIST [23] and the 2D Gaussian-bump benchmark adapted from [25]; both are cited at first use in Sec. 4.1 and 4.4, used under their public terms.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

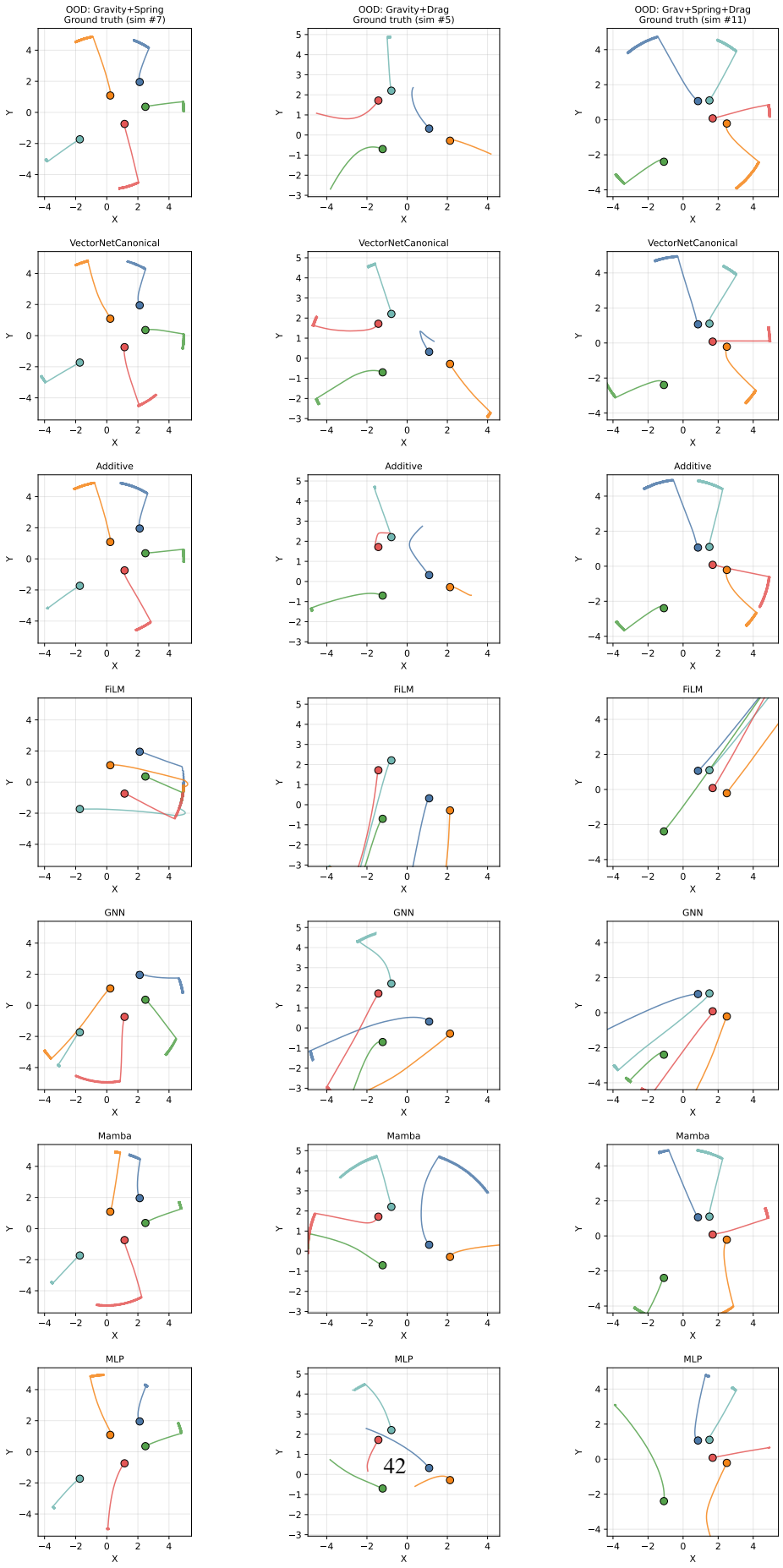
    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: This work's core method development did not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.


Trajectory rollouts -- 999 steps, dt=0.005
OOD: Gravity+Spring
Ground truth (sim #7)
OOD: Gravity+Drag
Ground truth (sim #5)
OOD: Grav+Spring+Drag
Ground truth (sim #11)
VectorNetCanonical
VectorNetCanonical
VectorNetCanonical
Additive
Additive
Additive
FiLM
FiLM
FiLM
GNN
GNN
GNN
Mamba
Mamba
Mamba
MLP
MLP
MLP
X
Y
42