Learning Options for Compositional Motor Control with Adapter Banks

Sreejan Kumar[1,2] **Marcelo G. Mattar**[2] * **Lea Duncker**[1]*

[1]Columbia University [2]New York University

Abstract

Learning flexible motor primitives is a hallmark of skilled motor control. Recent neuroscience theory proposes that motor primitives may be implemented as low-rank perturbations of a shared recurrent network, but leaves open how such a system is learned. We translate this principle into a novel architecture for learning motor skills end-to-end: a shared recurrent core modulated by a bank of residual adapters, each selected by a discrete latent code. Trained on closed-loop biomechanical control, the adapters develop emergent low-rank perturbations of the recurrent dynamics despite no architectural rank constraint, placing task representations in disparate subspaces of the shared core network. A simple high-level policy over the learned options, optimized while the whole network is frozen, sequences the low-rank adapters to produce novel out-of-distribution movements. We demonstrate the ability to generalize to novel motor sequences within the closed-loop control setting, improving on the generalization error of a task-input-conditioned multitask baseline by approximately sevenfold.

1 Introduction

In the movie *The Karate Kid*, how did Mr. Miyagi train his student to win a fighting tournament through seemingly unrelated chores like waxing cars and painting fences? The capacity to generate flexible behavior from a finite set of reusable primitives that are recombined and rescaled across new contexts is a hallmark of skilled motor control in humans [1, 2]. Reproducing this capacity in artificial agents has been a central goal of hierarchical and modular control: how can a learning system discover reusable building blocks of behavior, and flexibly recombine them to solve new problems [3, 4]?

To answer this question, we first ask: what form should these reusable building blocks take? In hierarchical reinforcement learning, reusable primitives are commonly formed as *options*: temporally extended actions, each with its own internal low-level policy, invoked by a high-level policy that decides which option to engage at each moment. Methods for discovering such skills learn both their individual low-level actions as well as how they can be recruited by the high-level policy [5–7]. In the brain, however, different movements draw on shared cortical circuitry [8], raising the question of how distinct, reusable operations can be implemented within a common network. One possibility is that a motor primitive corresponds to a modulation of that network's ongoing dynamics. Selecting a primitive would then configure how the network transforms its current state and sensory feedback into movement, allowing the same operation to act under different conditions.

Cortico–basal ganglia–thalamo–cortical circuits provide a candidate architecture for implementing this proposal. In the framework of Logiaco et al. [9], cortex supplies shared recurrent dynamics, while thalamocortical loops modify its effective connectivity. The basal ganglia acts as a selector of thalamic channels to determine which modifications are engaged, allowing distinct dynamical

*equally contributing senior authors

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

motifs to be recruited and sequenced within the same cortical network. This framework establishes how a bank of thalamic modulatory signals can generate composable motor operations. It leaves open, however, how the cortical network, the modulations acting upon it, and the control programs selecting them can be jointly learned to produce compositional motor control.

We address this learning problem with CABRA (Compositional Adapter Banks for Recurrent Architectures), a shared recurrent cortical network modulated by a bank of learned residual adapters representing thalamocortical loops, whose recruitment is determined by a basal ganglia controller. CABRA learns through complementary observation and execution passes. During observation, the cortical network processes an expert demonstration and supplies activity to the controller, which infers a sequence of adapters. During execution, the selected adapters modify cortical dynamics to generate behavior in closed loop, and execution error trains their effects. This couples learning which operations to recruit with learning how those operations act within a shared network. We implemented CABRA on the closed-loop musculoskeletal control setting of Lazzari and Saxena [10]. Three properties emerge from training. First, despite each adapter being initialized as a full-rank residual modulator, the adapters learn to apply emergent *low-rank* perturbations to the core network, recovering the structural assumption of Logiaco et al. [9]. Second, task representations are factored into disparate subspaces of a shared recurrent substrate, factoring the compositional structure of the task suite. Third, the trained adapter library supports out-of-distribution *compositional transfer*: a soft policy over the learned adapters, optimized while the rest of the network is frozen, sequences existing primitives into a novel trajectory the network was never trained on, outperforming a task-input-conditioned multitask baseline by approximately sevenfold on a novel two-petal movement. Together, these results show how learning within a cortico–basal ganglia–thalamic architecture can establish reusable motor operations that can provide the foundation to support more complex behavior over the same computational repertoire.

2 Low-rank Adapters via Thalamocortical Architectures in Neuroscience

Theoretical work has proposed an architecture for composing motor primitives into flexible sequences, inspired by the connectivity between motor cortex, thalamus and basal ganglia in the brain [9]. Here, motor primitives are implemented through an interplay of the recurrence in motor cortex, and cortex-thalamus-cortex projections (thalamocortical loops, Figure 1A). Distinct thalamic units participate in different thalamocortical loops, and activations of subsets of thalamic units via basal ganglia inputs can thus act as a low-rank perturbation of the effective recurrent dynamics in the motor cortex. This allows the system to generate complex movements by sequencing low-rank perturbations of the same shared cortical substrate.

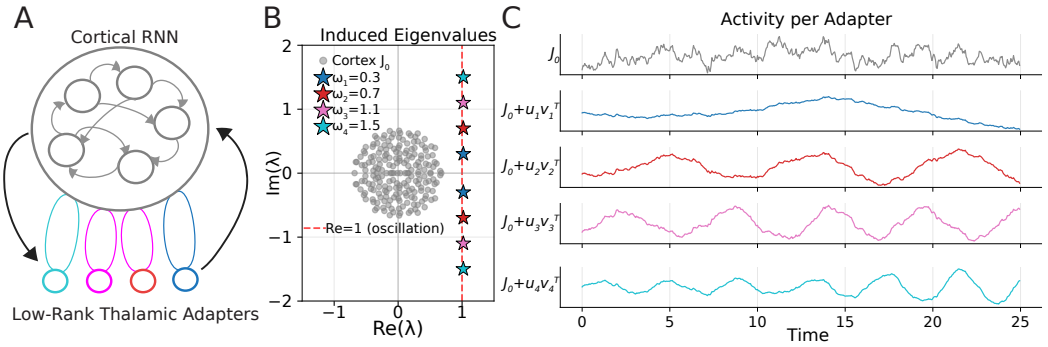

Figure 1: **Adapter Banks as Thalamocortical Loops (A)** Schematic of the framework of Logiaco et al. [9]: thalamic inputs $\mathbf{u}_k$ drive a cortical RNN with recurrence $\mathbf{J}_0$. A rank-one perturbation $\mathbf{u}_k\mathbf{v}_k^\top$ can induce oscillatory dynamics at a target frequency. **(B)** Eigenvalue placement. Gray: eigenvalues of the bare cortex $\mathbf{J}_0$. Stars: target locations on the oscillation line ($\mathrm{Re}(\lambda) = 1$) for four motifs at different frequencies ω_k. Each low-rank perturbation shifts a conjugate pair of eigenvalues to the corresponding target. **(C)** Cortical activity projected onto a one-dimensional readout. Each motif $J_0 + u_k v_k^T$ sustains a different oscillation at its target frequency ω_k, demonstrating that rank-one perturbations selectively recruit the cortical network to produce frequency-specific dynamics.

Let $\mathbf{h} \in \mathbb{R}^N$ denote the cortical state and $\mathbf{J}_0 \in \mathbb{R}^{N \times N}$ the recurrent connectivity in cortex. Each thalamocortical channel k consists of a corticothalamic readout $\mathbf{v}_k \in \mathbb{R}^N$ and a thalamocortical projection $\mathbf{u}_k \in \mathbb{R}^N$. The cortical dynamics evolve as

$$\tau \dot{\mathbf{h}} = -\mathbf{h} + \mathbf{J}_0 \mathbf{h} + \sum_k \mathbf{u}_k \mathbf{v}_k^\top \mathbf{h} = -\mathbf{h} + \left(\mathbf{J}_0 + \sum_k \mathbf{u}_k \mathbf{v}_k^\top \right) \mathbf{h}. \tag{1}$$

The thalamocortical loop therefore acts as a low-rank perturbation to the effective cortical dynamics. In the linear setting, Logiaco et al. [9] showed that this low-rank perturbation can be tuned to precisely modulate the eigenvalues of the effective dynamics (Figure 1B). It can therefore change the behavior of the network (e.g. oscillation frequency of activity patterns, Figure 1C) and introduce particular *dynamical motifs* that would not be expressible by $\mathbf{J}_0$ alone. This hence provides a mechanism for compositional motor control: the activity patterns needed to generate diverse movements can be constructed from a recurrent network by sequentially engaging different thalamocortical channels.

The framework in Logiaco et al. [9] showed how to design readout vectors $\mathbf{v}_k$ to achieve a particular change in eigenvalue of the effective dynamics and proposed that the selection mechanism can be implemented via disinhibition from basal ganglia circuits. However, they did not address how such an architecture could be *learned* more generally, and what mechanisms might govern motif *selection*. In the following section, we build on the conceptual framework of Logiaco et al. [9] to arrive at an algorithm for the learning and selection of motifs in more general, nonlinear settings.

3 Learning Adapters for Flexible Motor Control

We propose CABRA (Compositional Adapter Banks for Recurrent Architectures) as a framework for learning both which operations to recruit and what those operations do. We implement this through two complementary uses of a shared recurrent network: observing a demonstration to infer a control program, and acting under that program to reproduce the behavior (Figure 2B,C). These modes are paired within each training update.

3.1 Motor task and demonstrations

We adopt the closed-loop biomechanical control task from Lazzari and Saxena [10]. In this task, an RNN is required to control a two-link biomechanical arm with six muscles to generate a variety of movement types: straight reaches, clockwise and counterclockwise circular arcs, and sinusoidal trajectories. The RNN receives proprioceptive feedback (hand position, muscle length, muscle velocity) at each timestep. Movements come in two variants: *half* (start $\rightarrow$ target) and *full* (start $\rightarrow$ target $\rightarrow$ start). This yields ten distinct task conditions.

We use the model from Lazzari and Saxena [10] as a teacher network that can generate expert trajectories. The teacher network receives a one-hot *rule input* at every timestep, specifying which task is currently active (Figure 2A), alongside speed, go-cue, and target information, as well as proprioceptive feedback. As was demonstrated in Lazzari and Saxena [10], RNNs develop shared dynamical structure across movement types after training, allowing for reuse of the same computations across the different tasks. While this solution is compositional, the rule input supplies the network a particular organization of composable units rather than discovering them from learning. In the next section, we explore an alternative architecture with a bank of learnable adapters that can be flexibly selected to compose movements.

3.2 CABRA: Compositional Adapter Banks for Recurrent Architectures

To instantiate a brain-inspired principle for flexible motor control, we develop a general architecture which learns to observe expert demonstrations, segment them into learned discrete latent skills [6], and then use this decomposition to execute the same movement by selecting adapters to modulate the shared recurrent core. CABRA (Compositional Adapter Banks for Recurrent Architectures) comprises a shared recurrent core, a selector, and a bank of residual adapters. Their functional roles correspond to cortex, basal ganglia, and thalamocortical loops, respectively. The cortical core is a shared LSTM. The BG controller comprises readouts that predict adapter identities and transition boundaries from cortical activity. Each adapter is a learned nonlinear residual transformation of that activity, allowing the selected operation to depend on the current cortical state.

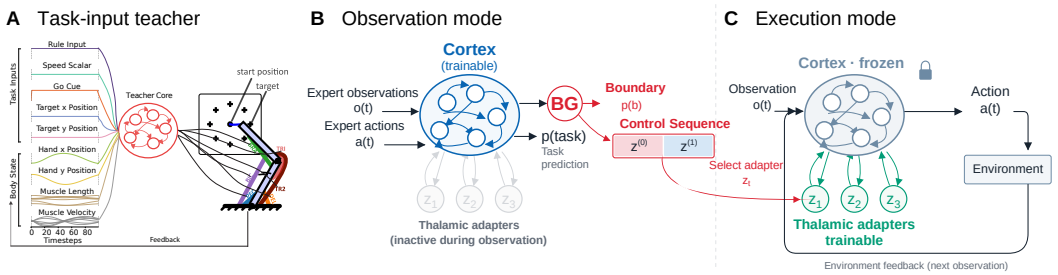

Figure 2: **An imitation learning framework to train flexible motor adapters (A)** We train a standard task-input-conditioned recurrent network controlling a two-link, six-muscle arm to supply expert demonstrations [10]. **(B)** In observation mode, CABRA's cortical network processes expert observations and actions without adapter modulation (note that CABRA's observations do not contain the rule input). Its activity supports a basal-ganglia-like (BG) selector that infers two adapter identities and their transition boundary, as well as an auxiliary task-identity prediction. **(C)** In execution mode, the inferred sequence recruits thalamic adapters that modulate the same cortical network to generate actions and receive environmental feedback. Execution error trains the adapters and readout.

Observation mode: inferring what to recruit. During observation, the shared cortical network processes the expert's sensory observations and muscle commands, without the rule input and with adapters inactive (Figure 2B). An auxiliary readout predicts the demonstrated task from cortical activity pooled over the first segment. This objective encourages representations of the identity of the observed behavior, motivated by evidence that cortical regions represent the goals of observed actions across changes in their immediate movement trajectories [11]. The BG controller reads cortical activity to infer a control program using the segmentation framework of CompILE [6]. We prescribe two segments, corresponding to extension and return, while learning their transition time and adapter identities. For each segment, the controller predicts a boundary distribution $p(b)$ over timesteps and a categorical distribution $p(z)$ over adapters. The complete program is inferred from the expert demonstration before execution.

During training, the code distribution is sampled with the straight-through Gumbel-softmax estimator, yielding a one-hot sample $z_{m,k} \in \{0, 1\}$ with $\sum_k z_{m,k} = 1$ in the forward pass so that exactly one adapter is active per segment. The boundary distribution is sampled with Gumbel-softmax at low temperature, inducing a soft segment assignment $\sigma_m(t) \in [0, 1]$ that concentrates near $\{0, 1\}$ except in the narrow window around the segment transition. At evaluation, both samples become hard via argmax. Half of the codes $(z_0, \ldots, z_4)$ are reserved for segment 0 (extension); the other half $(z_5, \ldots, z_9)$ for segment 1 (retraction). This phase split is implemented through hard masking of invalid logits and is the only structural prior we impose on the code identities themselves. The assignment of tasks to specific extension and retraction codes, and whether tasks reuse codes altogether, is up to the BG controller to learn.

Execution mode: learning the recruited operations. During execution, the inferred program recruits adapters that modulate the shared cortical network as it controls its own arm through sensory feedback (Figure 2C). The network receives neither expert muscle commands nor the rule input. Execution error directly optimizes the adapters, muscle readout, and execution-input projection, allowing the selected operations to acquire the effects needed for accurate behavior. This allocation is motivated by evidence that motor thalamus provides ongoing input essential to cortical movement dynamics, together with recent evidence that cerebellar-recipient motor thalamic neurons encode positional errors during reaching [12, 13]. These findings support a role for thalamocortical interactions in the ongoing regulation of movement.

The cortical recurrent weights are held fixed within the execution rollout and receive no direct execution-error gradient through that recurrence. However, the execution error can still shape the cortical recurrent weights indirectly through the BG controller's selection of the control program. Because adapter choices and their timing depend on cortical activity during observation, improving those choices also trains the cortical representations from which they are inferred. This constitutes

controller-mediated cortical learning, related to proposals that basal ganglia selection guides the development of cortical representations and associations [14, 15].

We impose no explicit rank constraint on the adapters. Each adapter (one per latent code) $\Delta_k : \mathbb{R}^N \to \mathbb{R}^N$ is a residual modulator of the core network's hidden state and any low-rank structure they develop in training emerges from the optimization itself, a property we examine quantitatively in Section 4. At each decoder timestep t, the core recurrent network produces a hidden state h_t from observations and the previous state, and the adapter perturbs it residually. Let $z^{(0)}, z^{(1)}$ denote the codes sampled by the encoders for both segments respectively, and let $\sigma(t) = \sum_{\tau \le t} b_\tau$ denote the cumulative probability the segment boundary has occured by time t. At decoder timestep t, the core network produces a hidden state h_t and the active adapters perturb it residually:

$$\tilde{h}_t = h_t + (1 - \sigma(t))\Delta_{z^{(0)}}(h_t) + \sigma(t)\Delta_{z^{(1)}}(h_t) \tag{2}$$

Because $\sigma(t)$ is sharply concentrated around the sampled boundary, the perturbation collapses to $\Delta_{z^{(0)}}(h_t)$ during the extension segment and $\Delta_{z^{(1)}}(h_t)$ during the retraction segment, with brief soft mixing only across the transition. The muscle readout reads from $\tilde{h}_t$, the arm advances one-step through differentiable physics, and $\tilde{h}_t$ is fed back to the core network as the next-step recurrent input. The decoder receives the same task inputs and closed-loop body-state feedback the teacher uses during training, but never the rule input. Removing the adapters or replacing them with a single shared modulator prevents the network from effectively learning (Supplementary Figure 9).

Training objectives. We train the full system end-to-end with a hand-trajectory L_1 loss against the teacher's trajectories, KL regularization on boundary and code distributions, and standard activity and muscle-effort regularizers following Lazzari and Saxena [10]. Two auxiliary losses improve training stability and sharpen code routing: a behavioral discriminator that predicts the active code from segment-pooled muscle commands (DIAYN, Eysenbach et al. 5), which reads only the six-dimensional muscle output and predicts the code that produced it; and an L_1 loss between student and teacher muscle excitations. Neither auxiliary loss directly determines the assignment of tasks to codes: the discriminator operates on muscle output rather than routing, and excitation matching shapes muscle commands rather than code logits. We show in Supplementary Figure 8 that both auxiliary losses can be ablated without losing the architectural transfer learning advantage.

We train CABRA only on compound trajectories (FullReach, FullCircleClk, FullCircleCClk, Figure-8, Figure-8 Inv) and never on the half-task primitives that compose them. Any decomposition into reusable half extension and retraction primitives must therefore arise from training rather than from being explicitly demonstrated as a half movement.

4 Results

4.1 Compound performance and meaningful code routing

We first verify that CABRA successfully reproduces the teacher's compound movements (Figure 3, bottom row). Trained only on compound trajectories and without ever being supplied the rule as input, CABRA traces the expert circles, figure-eights, and reaches with end-to-end hand L1 of approximately order 10^{-2} m on all five tasks. The reach is essentially exact; the figure-eights show closed-loop wobble deviations, but recover the general shape.

The more revealing analysis is the forced-code sweep (Figure 3, top row). For each compound task, we run the trained decoder under all 25 possible (extension code, retraction code) combinations and measure hand-trajectory error against the expert. If the discovered codes index meaningful primitives, forcing the model to use the wrong code should degrade behavior. If instead the codes are interchangeable, the heatmaps should be roughly uniform.

The pattern in the extension dimension shows specialized codes. Each task's natural extension code lies in a distinct row: Reach selects z_2, Circle CW selects z_0, Circle CCW selects z_3, Figure-8 selects z_1, and Figure-8 Inverse selects z_4. Forcing any other extension code raises error substantially. All five extension codes are populated, with a one-to-one assignment from tasks to codes. The training signal carries task-discriminative information (the encoder predicts task identity through an auxiliary readout, and the muscle output is matched to the teacher's task-specific commands) but does not

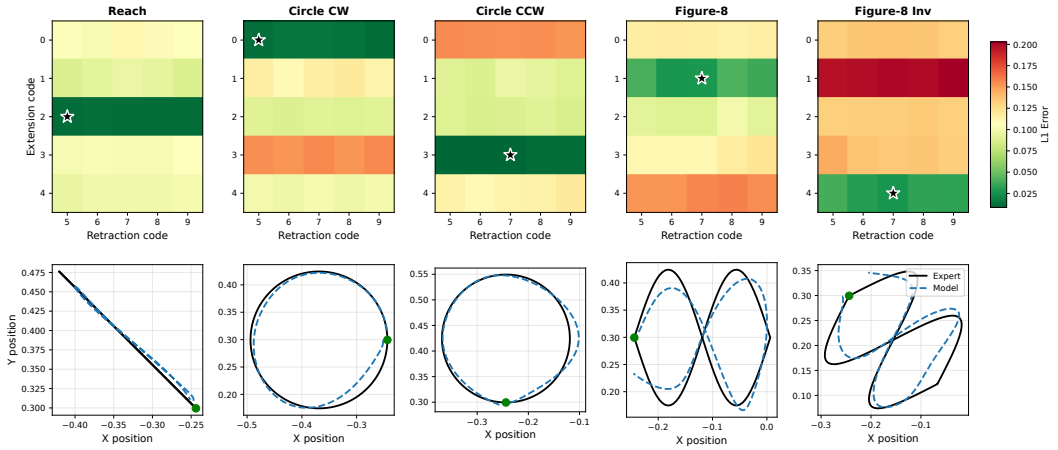

Figure 3: **CABRA execution of compound movements.** *Top row:* Hand-trajectory L1 error of the trained CABRA under all 25 forced (extension code, retraction code) pairs, for each of the five compound tasks. Rows index the five extension codes (z_0–z_4); columns index the five retraction codes (z_5–z_9). Stars mark the natural code pair selected by the encoder. *Bottom row:* Expert (black) and CABRA (blue, dashed) hand trajectories under the encoder's natural code choice. Green dots mark the start position.

specify the assignment from tasks to codes. The network is free to consolidate multiple tasks onto a single code or to populate the available codes one-to-one, and we observe the latter.

The retraction dimension is partially shared rather than one-to-one: Reach and Circle CW both select z_5 while Circle CCW, Figure-8, and Figure-8 Inverse all select z_7. Within a task's preferred extension code, swapping the retraction code from z_5 to z_9 produces only modest changes in trajectory error. This asymmetry between extensions and retractions tracks from the fact that the task prediction of the encoder is read out from the midpoint of the trial (specifically, the task readout is applied only to the first-segment encoding) and the kinematics of the task suite (several compound retractions converge to similar end-state trajectories).

4.2 Emergent Low-Rank Structure

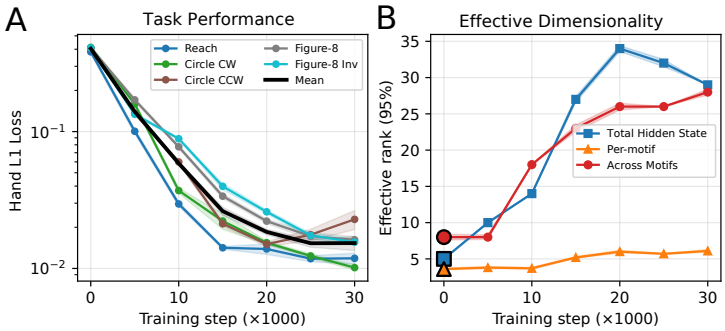

Figure 4: **Emergent low-rank structure in the adapter bank.** *(A)* Held-out hand L_1 per task and mean over training. *(B)* Effective rank (95% variance) of three quantities computed from closed-loop rollouts: the recurrent core's hidden-state manifold, the per-adapter perturbation subspace (averaged across adapters), and the union of all adapter perturbations. Bands show bootstrap uncertainty (standard deviation across resampled rank estimates); the individual-adapter curve averages adapter ranks. These are activity dimensions, not weight-matrix ranks.

We now ask whether the trained architecture exhibits the particular structural assumption that was made within Logiaco et al. [9]: motifs that are individually low-dimensional and jointly span the cortical state space. Recall that we impose no rank constraint on the motif adapters, because each

is initialized as a full-rank residual modulator with an unconstrained $H \to H$ map. Any low-rank structure they develop in training is therefore a property the architecture discovers, not one we imposed.

Figure 4A shows that all five compound tasks learn simultaneously. We characterize the dimensionality of the trained adapter bank's perturbation space relative to the dimensionality of the recurrent core's hidden-state manifold (Figure 4B). Under the core-focused analysis protocol (Supplementary Methods), the recurrent core's hidden-state activity reaches an effective rank of ~ 30 by end of training (out of the available $H = 256$ dimensions), indicating the core develops a structured low-dimensional manifold. Each individual adapter's perturbation subspace converges to an effective rank of only $\sim$ 5–6, which is well below both the core activity dimensionality and the available adapter dimensionality. And the union of all ten adapters' perturbations spans an effective rank of $\sim$ 28, nearly matching the core's full hidden-state manifold. Here, dimensionality refers to the expressed perturbations over sampled states, not the algebraic rank of adapter weight matrices. The distinctive result is the contrast between individually low-dimensional perturbations and their richer combined span within a learned recurrent network. Individual adapter perturbations remained lower-dimensional than core activity with smaller banks of six or eight adapters (Supplementary Figure 11).

4.3 Shared core, separated representations

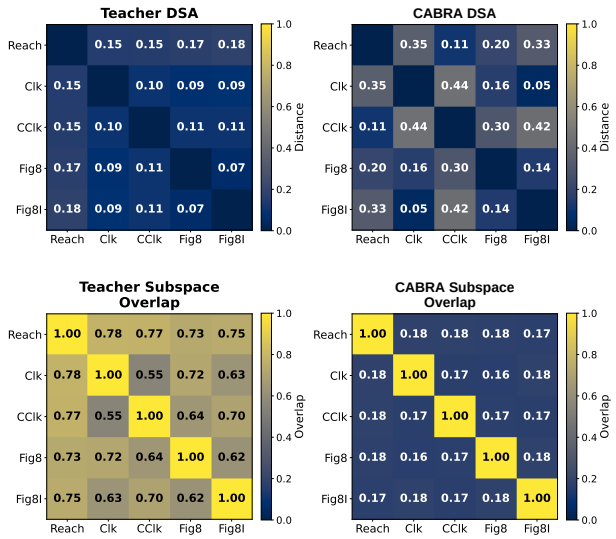

Figure 5: **Task-input baseline versus CABRA dynamics and representational geometry.** Pairwise dynamical similarity (DSA distance, top row; lower means more similar dynamics) and principal-subspace overlap (bottom row; mean squared cosine of principal angles between the dominant cortical activity subspaces, higher means more shared geometry) between the five compound tasks, computed for the teacher (left) and CABRA (right).

We now compare the dynamical and representational organization of CABRA to the teacher, a vanilla closed-loop multitask rule input-conditioned RNN. The two networks were trained on the same compound movements, although our network was only trained on compound movements while the teacher's training also included half extension-only movements. The teacher is a recurrent network that is modulated by a one-hot rule input, while CABRA is also a recurrent network but is modulated by learned adapters. We characterize each network along two complementary axes: *representational geometry* (how recurrent hidden state activity for different tasks occupies the state space, measured via principal-subspace overlap) and *dynamical structure* (how the underlying recurrent dynamics differ across tasks, measured via DSA). Figure 5 shows that this architectural difference produces a sharp dissociation along the representational axis, with corresponding structure along the dynamical axis.

Representational geometry. Following Lazzari and Saxena [10], we compute the principal-subspace overlap between pairs of tasks via principal-angle analysis. For each task A, we collect recurrent hidden state activity over the trajectory into a matrix $\mathbf{X}_A \in \mathbb{R}^{N \times T}$ ($N = 256$ units, T timesteps) and extract the top m principal components $\mathbf{P}_A \in \mathbb{R}^{N \times m}$ explaining $\geq 95\%$ of task variance. We use $m_{\text{teacher}} = 12$ and $m_{\text{CABRA}} = 25$, each matching its network's typical effective rank under the 95% variance threshold. The pairwise principal-subspace overlap between tasks A and B is then $\frac{1}{m}\sum_{i=1}^{m} \cos^2 \theta_i$ where $\{\cos \theta_i\}_{i=1}^{m} = \text{svd}\big(\mathbf{P}_A^\top \mathbf{P}_B\big)$ and θ_i are the principal angles between

the two subspaces. The teacher's pairwise overlap (Figure 5, bottom-left) is uniformly high, with off-diagonal mean 0.69. Task representations live in nearly the same recurrent subspace, consistent with the shared-manifold organization reported by Lazzari and Saxena [10]. CABRA's overlap (bottom-right) is uniformly low, with off-diagonal mean 0.18, less than $2\times$ its random baseline. Task representations in CABRA are highly separated in state space. This is the core empirical realization of the framework from Logiaco et al. [9]: a shared substrate modulated by discrete perturbations into minimally-overlapping subspaces. Removing the phase mask, so that every adapter could be selected for either extension or return, preserved both low-dimensional adapter perturbations and separated task representations (Supplementary Figure 12).

Dynamical structure. We assess the similarity of recurrent dynamics across tasks using Dynamical Similarity Analysis (DSA, Ostrow et al. 16), again following the protocol of Lazzari and Saxena [10]. The teacher's DSA matrix (Figure 5, top-left) is uniformly low, with off-diagonal mean 0.12: dynamics are similar across all task pairs. This is the within-manifold dynamical reuse that produces the teacher's single shared subspace. CABRA's DSA matrix (top-right) tells a different story. Off-diagonal mean is 0.25, materially higher than the teacher's, but the values are not uniform in that they are organized along behavioral axes. Pairs sharing rotational sense have low DSA distance (Circle CW vs. Figure-8 Inverse: 0.05; Circle CCW vs. Reach: 0.11), while rotation-opposite pairs separate sharply (Circle CW vs. Circle CCW: 0.44; Circle CCW vs. Figure-8 Inverse: 0.42). The adapters do not produce small residual perturbations of a strictly common dynamical system, but rather genuinely distinct dynamical systems organized by behavioral kinematics.

These two findings are complementary. The teacher reuses both dynamics and representations: its compositional structure lives in a single shared subspace, modulated by explicit rule inputs. CABRA factors the problem differently. Its recurrent *substrate* is shared: a single LSTM, the same recurrence, the same input projection, and the same readout. However, the adapter perturbations introduce structured dynamical differences and place task representations in highly separated subspaces. Where the teacher's compositional structure is tangled within a single manifold, CABRA's is geometrically factored.

4.4 Compositional transfer to a novel two-petal trajectory

The previous section established that CABRA's task representations occupy highly separable subspaces while the teacher's collapse into a single shared manifold. If this geometric factoring is what enables compositional reuse, then mixing primitives to compose a novel trajectory should be substantially easier in CABRA. We test this prediction directly.

We construct a synthetic four-segment trajectory (Figure 6A) that splices primitive movements in pairings neither network was trained on. Segment 1 is a straight extension followed by Segment 2's curved retraction, which contains a kinematic discontinuity at the target where the network must transition from a straight-line motor program to a curved-arc one. Segment 3 is a curved extension followed by Segment 4's straight retraction and represents the same discontinuity in the opposite direction. Neither (straight ext, curved ret) nor (curved ext, straight ret) appeared in either network's training data. We freeze both networks and optimize a 10-dimensional soft policy over the network's task interface (only learning a $T \times 10$ matrix): rule-input weights for the teacher and adapter-code weights for CABRA. We use gradient descent on the hand-trajectory L_1 to the two-petal target (Figure 6B). Both networks use the same optimizer, the same target, the same initialization heuristic, and the same evaluation procedure. The only thing that differs is the frozen substrate.

CABRA tracks the target to within $L_1 = 0.005$ while the teacher plateaus at $L_1 \sim 0.04$, approximately sevenfold higher error (Figure 6B,C). CABRA's optimized policy (Figure 6D, bottom) shows that mixing concentrates around the segment transitions, but extends into within-segment timesteps as well (most prominently during Segment 3). CABRA also produces qualitatively distinct retractions across the two compositions: z_7 (producing a curved retraction) follows the straight Segment-1 extension, while z_5 (straight) follows the curved Segment-3 extension. Although forced-code analysis (Figure 3) showed that the choice of retraction code mattered little for in-distribution compound performance, in this OOD setting CABRA's selection policy meaningfully differentiates retraction primitives to match the trajectory's demands. The teacher's policy (Figure 6D, top), in principle, could mix multiple rule inputs to compose the target (and a few timesteps show such co-activation), but the transfer optimization consequently gravitates toward solutions where one rule dominates at a time, leaving the teacher unable to manage transitions within novel splice-compositions. The

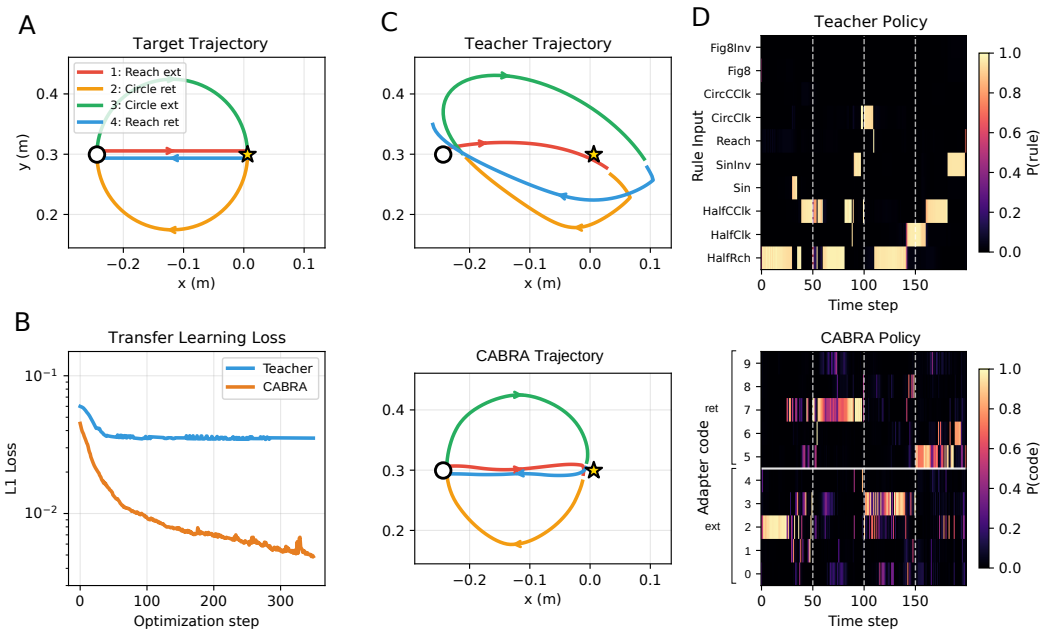

Figure 6: **Out-of-distribution compositional transfer.** *(A)* The two-petal target trajectory, composed of four movement segments stitched together in a novel combination. *(B)* Hand L_1 during transfer optimization, in which both networks are frozen and a 10-dimensional soft policy over the network's task interface is optimized for 350 steps via gradient descent. Final teacher $L_1 \approx 0.04$; final CABRA $L_1 \approx 0.005$. *(C)* Final teacher (top) and CABRA (bottom) hand trajectories under the optimized policies. Open circle marks the home position; star marks the reach target. *(D)* Final policies. Top: teacher rule-input weights over time. Bottom: CABRA adapter-code weights. Adapter IDs are consistent across rows; brackets mark the extension and return training pools, while transfer optimization can recruit the full bank.

architectural advantage holds on a second more difficult out-of-distribution shape (Supplementary Figure 7). CABRA's advantage is preserved when both networks are restricted to hard discrete selection at each timestep ($2.5\times$ gap in best error, $2.2\times$ over the final 50 updates; Supplementary Figure 10).

We also evaluated CABRA on D4RL Kitchen, a standard benchmark used in hierarchical reinforcement learning in which a 3D robotic arm completes household manipulation tasks, where a reward-trained policy over frozen adapters achieved higher early-training reward and comparable late-training performance to the skill-learning method SPiRL [17] (Supplementary Figure 13).

5 Discussion

We investigated how reusable motor operations can be learned within a shared recurrent network whose dynamics are selectively modulated. CABRA couples learning to infer control programs from demonstrations with learning the effects of the operations those programs recruit. In closed-loop musculoskeletal control, this procedure establishes a repertoire that supports both trained movements and an untrained movement acquired through new control over the frozen network. These results extend a proposed circuit mechanism for composing motor motifs into a more general learning architecture, showing how operations acquired in particular behavioral contexts can remain useful beyond those contexts.

An important extension of the framework of Logiaco et al. [9] is that shared cortical dynamics and the primitives that recruit them are learned together. Their theory showed how structured thalamocortical loops can configure a fixed, randomly connected recurrent network to generate distinct motor motifs. In CABRA, the recurrent network itself acquires structure during learning, motivated by the rich movement-generating dynamics observed in motor cortex [8]. Learning therefore shapes both the

dynamical resources shared across movements and the adapter operations that selectively recruit them for individual movements. The resulting movements occupy differentiated cortical subspaces while retaining shared dynamical structure organized by behavioral kinematics (Figure 5). Distinct motor primitives can thus draw on common learned dynamics while remaining separately accessible to control. Their separation does not require independent recurrent networks, and their sharing does not require identical cortical activity. This organization emerges as the shared substrate and its selectively recruited modulations develop together.

Related approaches offer complementary accounts of how shared computation can support flexible behavior. Costacurta et al. [18] configure recurrent computation by modulating gains on a learned low-rank connectivity basis. CABRA instead learns nonlinear residual operations around a full-width recurrent core, with low-dimensional perturbations emerging through training rather than being imposed architecturally. Work by Shan et al. [19] and Bakermans et al. [20] likewise emphasizes how established computational structure can support new tasks through changes in control or inference. CABRA operates on a similar principle, but learns this structure instead. The differentiated task subspaces also connect to Duncker et al. [4], who showed how the organization of recurrent dynamics into orthogonal subspaces can mitigate interference during sequential continual learning. Our transfer experiment addresses a complementary question: whether new behavior can be acquired by changing control while preserving the learned repertoire. Although differentiated representations and successful reuse coexist here, establishing which features of that organization cause transfer would require more targeted interventions.

This account also links two perspectives on basal ganglia function: selecting actions [21] and influencing the detailed kinematics of learned movements [22]. When the selected object is an operation on ongoing cortical dynamics, a discrete selection can have continuously evolving effects on movement. Its consequences depend on the cortical state and sensory feedback, rather than specifying a fixed trajectory independently of the network. In the novel-movement experiment, sequencing individual adapters retained the movement's broad structure, while graded combinations improved trajectory accuracy (Figure 6; Supplementary Figure 10). Selection and kinematic refinement may therefore be complementary uses of the same interface, CABRA being a concrete computational instantiation of that interface.

An important direction for future work is to learn more of the structure currently supplied by the training setup. In our main experiments, two segments reflect the extension–return organization of the demonstrations, while the adapter-bank size sets the capacity of the available repertoire without prescribing task-to-adapter assignments. These choices influence the granularity of the learned operations: fewer adapters encourage sharing, whereas larger banks permit greater specialization but may also introduce redundant operations. Our reduced-capacity experiments retain low-dimensional adapter perturbations while increasing overlap between task representations, illustrating this tradeoff (Supplementary Figure 11). A natural extension is therefore to infer how many operations a behavioral repertoire requires, together with when those operations begin and end. The learning objectives also contain dataset-specific and engineering choices: task-identity supervision emphasizes information distinguishing the demonstrated movements, while identity and muscle-excitation objectives encourage adapter differentiation and accurate execution. Our Kitchen experiments demonstrate that CABRA can learn useful operations with less semantic supervision and task-specific structure: they omit task-identity supervision and the extension–return phase mask, instead learning from fixed-duration demonstration windows (Supplementary Figure 13). Learning useful decompositions with less semantic supervision and fewer structural assumptions is a valuable direction for future work to strengthen the generality of this architecture.

The proposed mechanism suggests examining motor learning through the joint evolution of cortical dynamics and the signals that recruit them. Simultaneous cortical, thalamic, and basal ganglia recordings could test whether recurring control patterns accompany reproducible changes in cortical dynamics across movements. Perturbations could further distinguish disruption of an operation's execution from disruption of its selection, while neural-data constraints could help distinguish task-equivalent model solutions [23]. Such experiments can use real data to test and ground the central implication of our theory, which is that learning a movement can establish more than the ability to reproduce it: it can establish an operation on shared neural circuitry that subsequent control can recruit for a different purpose.

References

[1] Tamar Flash and Binyamin Hochner. Motor primitives in vertebrates and invertebrates. *Current opinion in neurobiology*, 15(6):660–666, 2005.

[2] James B Heald, Máté Lengyel, and Daniel M Wolpert. Contextual inference underlies the learning of sensorimotor repertoires. *Nature*, 600(7889):489–493, 2021.

[3] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75: 1401–1476, 2022.

[4] Lea Duncker, Laura Driscoll, Krishna V Shenoy, Maneesh Sahani, and David Sussillo. Organizing recurrent network dynamics by task-computation to enable continual learning. *Advances in neural information processing systems*, 33:14387–14397, 2020.

[5] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[6] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pages 3418–3428. PMLR, 2019.

[7] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[8] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Justin D Foster, Paul Nuyujukian, Stephen I Ryu, and Krishna V Shenoy. Neural population dynamics during reaching. *Nature*, 487(7405):51–56, 2012.

[9] Laureline Logiaco, LF Abbott, and Sean Escola. Thalamic control of cortical dynamics in a model of flexible motor sequencing. *Cell reports*, 35(9), 2021.

[10] John Lazzari and Shreya Saxena. Multitasking recurrent networks utilize compositional strategies for control of movement. *bioRxiv*, pages 2025–09, 2025.

[11] AF Hamilton and Scott T Grafton. The motor hierarchy: from kinematics to goals and intentions. *Sensorimotor foundations of higher cognition*, 22:381–408, 2007.

[12] Rex Tien, Jonathan Platt, Madelyn Mendlen, Drew Kern, Steven Ojemann, John Thompson, and Daniel Kramer. Neurons in human motor thalamus encode reach kinematics and positional errors related to braking. *Research Square*, pages rs–3, 2025.

[13] Britton A Sauerbrei, Jian-Zhong Guo, Jeremy D Cohen, Matteo Mischiati, Wendy Guo, Mayank Kabra, Nakul Verma, Brett Mensh, Kristin Branson, and Adam W Hantman. Cortical pattern generation during dexterous movement is input-driven. *Nature*, 577(7790):386–391, 2020.

[14] Anitha Pasupathy and Earl K Miller. Different time courses of learning-related activity in the prefrontal cortex and striatum. *Nature*, 433(7028):873–876, 2005.

[15] F Gregory Ashby, John M Ennis, and Brian J Spiering. A neurobiological theory of automaticity in perceptual categorization. *Psychological review*, 114(3):632, 2007.

[16] Mitchell Ostrow, Adam Eisen, Leo Kozachkov, and Ila Fiete. Beyond geometry: Comparing the temporal structure of computation in neural circuits with dynamical similarity analysis. *Advances in Neural Information Processing Systems*, 36:33824–33837, 2023.

[17] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 188–204, 2021.

[18] Julia C Costacurta, Shaunak Bhandarkar, David Zoltowski, and Scott W Linderman. Structured flexibility in recurrent neural networks via neuromodulation. *Advances in Neural Information Processing Systems*, 37:1954–1972, 2024.

[19] Haozhe Shan, Sun Minni, and Lea Duncker. Separating the what and how of compositional computation to enable reuse and continual learning. *arXiv preprint arXiv:2510.20709*, 2025.

[20] Jacob JW Bakermans, Pablo Tano, Reidar Riveland, Charles Findling, and Alexandre Pouget. Compositional meta-learning through probabilistic task inference. *arXiv preprint arXiv:2510.01858*, 2025.

[21] Jonathan W Mink. The basal ganglia: focused selection and inhibition of competing motor programs. *Progress in neurobiology*, 50(4):381–425, 1996.

[22] Ashesh K Dhawale, Steffen BE Wolff, Raymond Ko, and Bence P Ölveczky. The basal ganglia control the detailed kinematics of learned motor skills. *Nature neuroscience*, 24(9):1256–1269, 2021.

[23] Adrian Valente, Jonathan W. Pillow, and Srdjan Ostojic. Extracting computational mechanisms from neural data using low-rank RNNs. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[24] Olivier Codol, Jonathan A Michaels, Mehrdad Kashefi, J Andrew Pruszynski, and Paul L Gribble. Motornet, a python toolbox for controlling differentiable biomechanical effectors with artificial neural networks. *Elife*, 12:RP88591, 2024.

A Supplementary Methods

A.1 Architecture details

CABRA uses one shared LSTM core in observation and execution modes, with separate input projections, selection heads, a muscle readout, and a bank of adapters. In the implementation, the observation pathway is termed the encoder and the execution pathway the decoder. The core and adapter hidden dimension is $H = 256$.

Shared LSTM core. A single LSTM of width $H = 256$ is used both for encoder and decoder forward passes. Recurrent parameters are detached during the decoder pass, preventing direct execution-path updates to those weights. Hand-trajectory error still reaches the shared core indirectly through the encoder-produced control variables, as well as updating the decoder input projection, adapters, and muscle readout.

Input projections. The encoder and decoder receive different observation subsets and use separate input projection MLPs (each a two-layer MLP with ReLU activation, hidden width $H{=}256$):

- Encoder input: 24-dim concatenation of the 18-dim sliced observation (full observation with the 10-dim rule input stripped) and the 6-dim teacher muscle command.
- Decoder input: 18-dim sliced observation only (no muscle command, no rule input).

Observation-mode selection and task heads. Three heads read from the encoder LSTM hidden state. The boundary and code heads are two-layer MLPs ($H \to H \to \cdot$) with a ReLU between layers; the task head is also a two-layer MLP with ReLU.

- *Boundary head*: outputs a scalar logit per timestep, softmaxed across time within each segment to yield $p(b)$.
- *Code head*: outputs $K = 10$ logits per segment. Sampled with the straight-through Gumbel-softmax estimator at temperature $\tau_z = 0.1$, yielding a one-hot $z_{m,k}$ in the forward pass with gradients flowing through the soft distribution.
- *Task head* (auxiliary): two-layer MLP ($H \to 64 \to 5$) reading from the encoder's first-segment pooled hidden state, predicting task identity. Loss propagates only through the encoder.

Adapter bank. Each of the $K = 10$ adapters is a two-layer nonlinear residual modulator:

$$\Delta_k(\mathbf{h}) = \mathbf{W}_k^{\text{out}} \tanh\big(\mathbf{W}_k^{\text{in}}\mathbf{h}\big)\,,$$

where $\mathbf{W}_k^{\text{in}}, \mathbf{W}_k^{\text{out}} \in \mathbb{R}^{H\times H}$. No bias terms. The output weights $\mathbf{W}_k^{\text{out}}$ are scaled by a factor of 2 at initialization to produce non-negligible early perturbations and accelerate adapter learning. No rank constraint is imposed on Δ_k in the architecture; the empirical rank of the learned adapter perturbation in the cortical hidden state is analyzed in Section 4.

Phase-split mask. Codes are partitioned by movement phase: codes $z_0, \ldots, z_{K/2-1}$ are reserved for segment 0 (extension) and $z_{K/2}, \ldots, z_{K-1}$ for segment 1 (retraction). The phase split is implemented via hard masking of the invalid logits before sampling and is the only structural prior we impose on code identities; the assignment of tasks to specific codes within each phase emerges from training.

From segment selections to a control sequence. For segment $m \in \{1, 2\}$, let $z_{m,k}$ denote the sampled recruitment of adapter k, indexed from 0 to $K-1$. If $\sigma(t) = \sum_{\tau\le t} b_\tau$ is the cumulative probability that the inferred boundary has occurred, the time-dependent weights in Equation 2 are $z_{t,k} = (1-\sigma(t))z_{1,k} + \sigma(t)z_{2,k}$. Thus discrete segment identities and a soft boundary produce the full control sequence; at evaluation, identities and boundary are hardened by argmax.

Muscle readout. A linear layer maps the (adapter-perturbed) hidden state $\tilde{\mathbf{h}}_t$ to a 6-dimensional pre-activation, passed through a sigmoid to produce the muscle excitation vector supplied to the differentiable arm.

DIAYN discriminator. The DIAYN [5] discriminator is implemented in `binned_muscle` mode: muscle commands within each segment are pooled into $n_{\text{bins}} = 3$ temporal bins and concatenated to form a $3 \times 6 = 18$-dim feature, fed to a two-layer MLP ($18 \to 64 \to K/2$) that predicts the active code restricted to the segment's valid phase subset. Separate discriminators are used for extension and retraction segments. The discriminator reads only muscle output, never cortical state.

A.2 Training procedure

Optimization. Training uses Adam with learning rate 10^{-4}, batch size 32, gradient clipping at norm 1.0, and no weight decay. The full system is trained end-to-end for 30,000 iterations. Each training step pairs an observation pass and an execution pass through the shared LSTM.

Closed-loop rollouts. At each training iteration, the encoder forward pass reads the full teacher demonstration (observation and muscle command) and produces $p(b)$, $p(z)$, and the task-identity logits. The decoder forward pass uses the sampled $z_{m,k}$ and $\sigma_m(t)$ to roll out the closed-loop arm trajectory using only body-state feedback. The hand-trajectory L_1 loss is computed between CABRA's closed-loop rollout at the current training step and the teacher's pre-recorded hand trajectory for the same trial. Both trajectories are produced by closed-loop rollouts through the same differentiable arm: the teacher's was recorded once during demonstration generation; CABRA's is produced fresh each training step from its current parameters.

Gradient paths between observation and execution. The decoder uses detached copies of the shared recurrent parameters. Execution error cannot update those parameters through the execution recurrence, but can update them through the encoder that produced the sampled codes and boundaries. Detachment therefore separates gradient pathways; it does not insulate the shared core from all behavioral error signals.

Loss composition. The total loss decomposes into three components corresponding to the encoder's segmentation/inference objective, the decoder's closed-loop control objective, and a set of auxiliary losses:

$$\begin{aligned}
\mathcal{L}_{\text{enc}} &= \beta_z \mathcal{L}_{\text{KL}_z} + \beta_b \mathcal{L}_{\text{KL}_b} \\
\mathcal{L}_{\text{dec}} &= \mathcal{L}_{\text{hand}} + \lambda_{\text{rate}}\mathcal{L}_{\text{rate}} + \lambda_{\text{musc}}\mathcal{L}_{\text{musc}} \\
\mathcal{L}_{\text{aux}} &= \beta_{\text{DIAYN}}\mathcal{L}_{\text{DIAYN}} + \lambda_{\text{exc}}\mathcal{L}_{\text{exc}} + \lambda_{\text{task}}\mathcal{L}_{\text{task}} \\
\mathcal{L} &= \mathcal{L}_{\text{enc}} + \mathcal{L}_{\text{dec}} + \mathcal{L}_{\text{aux}}
\end{aligned}$$

The encoder objective $\mathcal{L}_{\text{enc}}$ regularizes the boundary and code distributions toward priors: $\mathcal{L}_{\text{KL}_z}$ is the KL between the encoder's per-segment code distribution and a uniform prior over the valid code subset; $\mathcal{L}_{\text{KL}_b}$ is the KL between the boundary distribution and a per-sample Poisson prior centered on the trial midpoint.

The decoder objective $\mathcal{L}_{\text{dec}}$ shapes the closed-loop motor output: $\mathcal{L}_{\text{hand}}$ is the per-timestep L_1 between CABRA and teacher hand trajectories averaged over the trial; $\mathcal{L}_{\text{rate}}$ and $\mathcal{L}_{\text{musc}}$ are L_1 regularizers on decoder hidden activity and muscle excitations following Lazzari and Saxena [10].

The auxiliary losses $\mathcal{L}_{\text{aux}}$ provide additional learning signal: $\mathcal{L}_{\text{DIAYN}}$ is the cross-entropy of the discriminator predicting the active code from segment-pooled muscle commands; $\mathcal{L}_{\text{exc}}$ is an L_1 loss between CABRA and teacher muscle excitations; $\mathcal{L}_{\text{task}}$ is the cross-entropy of the task-identity readout on the encoder's first-segment representation. The DIAYN and excitation-matching losses are ablated in Figure 8; task-identity supervision remains in those motor ablations.

Loss weights for the headline configuration are $\beta_z = 0.3$, $\beta_b = 0.1$, $\beta_{\text{DIAYN}} = 1.0$, $\lambda_{\text{rate}} = 10^{-3}$, $\lambda_{\text{musc}} = 10^{-2}$, $\lambda_{\text{exc}} = 0.5$, $\lambda_{\text{task}} = 0.5$.

Each training run was performed on a single NVIDIA GeForce GTX 1080 Ti GPU and took approximately 48 hours of wall-clock time to complete 30,000 iterations.

A.3 Task and data details

Biomechanical environment. We use the `motornet` [24] `RigidTendonArm26` effector with the `MujocoHillMuscle` model: a two-link arm with six muscles (two per joint plus two biarticular). Arm parameters match those used by Lazzari and Saxena [10].

Compound tasks. CABRA is trained on five compound trajectories (each comprising an extension followed by a retraction):

- **FullReach**: straight extension to target, then straight retraction home.
- **FullCircleClk**: clockwise half-circle outward, clockwise half-circle home.
- **FullCircleCClk**: counter-clockwise half-circle outward, counter-clockwise home.
- **Figure8**: half of a figure-8 outward, second half on return.
- **Figure8Inv**: figure-8 with inverted phase.

For each task we sample 32 reach conditions (target locations) per training batch. Each trial is a 200-timestep movement.

Demonstration information. The encoder sees the teacher's realized muscle commands together with sensory observations, whereas CABRA in execution mode generates its own commands and receives its own body-state feedback.

Half-task primitives are unseen. Although each compound trajectory consists of an extension followed by a retraction, we never expose CABRA to the half-task primitives in isolation. Any decomposition into reusable extension and retraction codes must arise from segmenting compound demonstrations rather than from explicitly demonstrated primitives.

A.4 Analysis methodology

Effective rank. For each adapter k, we collect the cortical perturbation $\Delta_k(\mathbf{h}_t)$ over a held-out task suite and compute the effective rank as the number of singular values needed to explain $\geq 95\%$ of the cumulative variance. Uncertainty is estimated from the standard deviation of rank estimates across $n_{\text{boot}} = 10$ bootstrap resamples; individual-adapter estimates are then averaged. For this core-focused assay, the unmodulated core state is carried into the next recurrent update, while adapter-modulated output still drives the arm and its feedback. This differs from native execution, which carries the modulated state forward. The core, individual perturbations, and pooled perturbations are measured under this common assay.

Principal-subspace overlap. For each task A, we collect cortical hidden state activity over the trajectory into $\mathbf{X}_A \in \mathbb{R}^{N\times T}$ ($N = 256$, T timesteps) and extract the top m principal components $\mathbf{P}_A \in \mathbb{R}^{N\times m}$ explaining $\geq 95\%$ of task variance. We use $m_{\text{teacher}} = 12$ and $m_{\text{CABRA}} = 25$, each matching its network's effective rank under the 95% variance threshold. The pairwise principal-subspace overlap between tasks A and B is

$$\text{overlap}(A, B) = \frac{1}{m}\sum_{i=1}^{m} \cos^2\theta_i, \quad \{\cos\theta_i\}_{i=1}^{m} = \text{svd}\big(\mathbf{P}_A^\top \mathbf{P}_B\big)\,,$$

where θ_i are the principal angles between the two subspaces. The chance baseline is the expected overlap between two random orthonormal m-dimensional subspaces in $\mathbb{R}^N$, computed via QR decomposition of Gaussian matrices (~ 0.05 for $m = 12$, ~ 0.10 for $m = 25$). Trajectories used for this analysis are pooled across 128 trials per task (32 reach conditions $\times 4$ repeats).

Dynamical Similarity Analysis. We follow the protocol of Lazzari and Saxena [10], computing DSA [16] pairwise between tasks. For each task we collect cortical activity into a tensor over conditions, trials, timesteps, and units, PCA-reduce to the network's effective rank ($m_{\text{teacher}} = 12$, $m_{\text{CABRA}} = 25$), construct delay-embedded Hankel tensors with lag $p = 90$, and fit reduced-rank Dynamic Mode Decomposition (DMD) matrices $\mathbf{A}_A, \mathbf{A}_B$ via Hankel Alternative View of Koopman (HAVOK) at rank $r = 150$. The DSA distance between tasks is the Procrustes distance under similarity transformation:

$$d(\mathbf{A}_A, \mathbf{A}_B) = \min_{\mathbf{C}\in O(r)} \|\mathbf{A}_A - \mathbf{C}\,\mathbf{A}_B\,\mathbf{C}^{-1}\|_2,$$

computed using the `DSA` package with `score_method="euclidean"`. All DSA hyperparameters match Lazzari and Saxena [10].

Transfer optimization. For both teacher and CABRA, we freeze the network and optimize a K-dimensional soft policy over the network's task interface (rule-input weights for the teacher, adapter-code weights for CABRA). Each policy is a learnable matrix of logits over time, softmaxed at temperature $\tau = 0.5$ to produce a per-timestep distribution over the $K = 10$ task interfaces. We optimize hand-trajectory L_1 loss against the two-petal target using Adam with learning rate 0.1 for 350 iterations. Both networks use identical optimizer settings, target trajectory, and policy initialization; the only difference is the frozen substrate. The butterfly trajectory uses 600 iterations at the same learning rate to accommodate its more complex shape. The softmax temperature in principle allows either policy to mix multiple task interfaces simultaneously at any timestep; differences in how soft each network's optimized policy ends up therefore reflect each network's underlying representational structure rather than any architectural asymmetry in the optimization itself.

Discrete selection and mixing. Code sampling, boundary sampling, and transfer optimization have distinct roles. Motor training uses straight-through discrete codes and a soft boundary assignment; evaluation hardens the inferred program. Novel-trajectory optimization instead permits graded adapter mixtures throughout the movement, not only at boundaries. The hard-selection control uses straight-through one-hot execution under the same transfer objective (Figure 10); it retains a transfer advantage at reduced accuracy. We did not perform a temperature sweep.

Forced-code analysis. For the qualitative inspection of code-task assignment (Figure 3), we run CABRA in closed loop while overriding the encoder's code distribution with a fixed one-hot code per segment. The resulting hand trajectory shows what behavior each adapter Δ_k produces in isolation, factored away from the encoder's learned code-inference policy.

A.5 Kitchen skill learning and reinforcement learning

We used demonstration trajectories from the Kitchen manipulation environment with 60-dimensional state observations and nine-dimensional actions. CABRA learned to predict actions over ten-step demonstration windows using a shared recurrent core of width 256 and ten full-width residual adapters. The demonstration encoder inferred a categorical skill distribution, and the decoder predicted actions conditioned on observed states and the inferred skill. Unlike the differentiable MotorNet rollout objective, this stage used action reconstruction on demonstration states. No task-identity classifier or two-phase mask was used, and window duration was fixed rather than inferred.

The offline objective combined action mean-squared error with aggregate code-distribution KL regularization toward a uniform distribution (weight 0.3), posterior-entropy regularization (0.01), a state-conditioned prior-matching loss (0.1), a skill-discrimination objective (1), and adapter-output and core-activity penalties (10^{-5} each). The prior predicted the inferred skill from the window's initial state, with the encoder posterior detached in the prior-matching target. Training used Adam at learning rate 10^{-4}, batch size 128, for 100 epochs.

For downstream control, we followed the skill-prior-regularized hierarchical reinforcement-learning setup of SPiRL [17]. Low-level networks remained frozen. A high-level policy selected a skill every ten environment steps and learned from task reward with KL regularization toward the frozen state-conditioned prior. CABRA used a categorical policy over adapters, initialized from the learned prior with a zero-initialized residual policy, and a discrete soft actor-critic update with exact categorical KL and target divergence 0.5. The SPiRL comparison used its continuous latent-skill policy and corresponding skill prior. Thus both methods use prior-regularized hierarchical control, but differ in skill parameterization and the associated policy implementation. Figure 13 shows the first 5,000 training episodes. Episode reward counts completed subtasks, up to four. Curves use nonoverlapping 100-episode bins and bands are the within-bin standard errors.

B Supplementary Figures

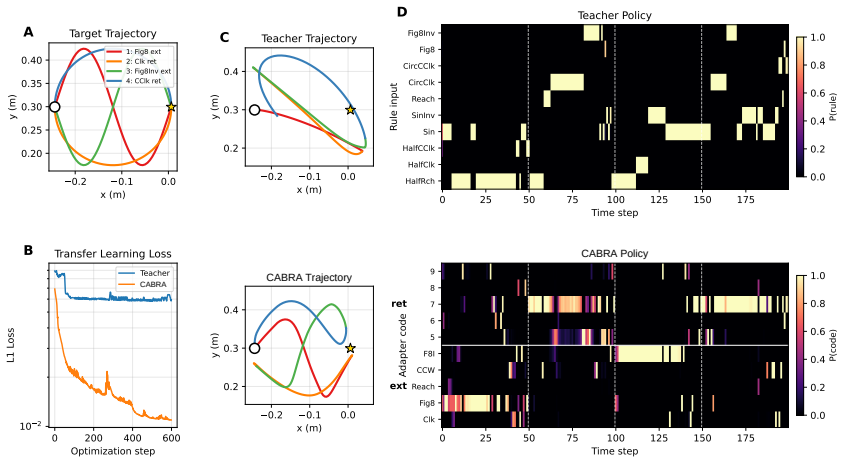

Figure 7: **Compositional transfer to a butterfly trajectory.** A second out-of-distribution test using a four-segment butterfly composed of (Figure-8 extension, Clockwise retraction, Figure-8 Inverse extension, Counter-clockwise retraction); none of these pairings appeared in either network's training data. *(A)* The butterfly target trajectory, with each 50-timestep segment shown in a separate color. Open circle marks the home position; star marks the reach target. *(B)* Hand L_1 during transfer optimization. Both networks are frozen and a 10-dimensional soft policy over the network's task interface is optimized for 600 steps. CABRA reaches hand $L_1 \approx 0.011$, while the teacher plateaus at ≈ 0.058, a $\sim 5\times$ gap. *(C)* Final teacher (top) and CABRA (bottom) hand trajectories. CABRA recovers the butterfly's figure-8-like crossing structure; the teacher's trajectory is qualitatively distorted. *(D)* Final policies. Top: teacher rule-input weights over time. Bottom: CABRA adapter-code weights.

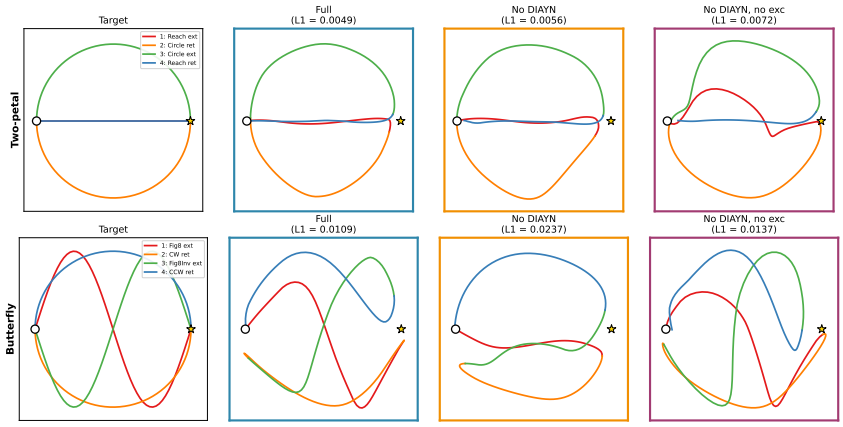

Figure 8: **Ablation of auxiliary supervision components across two OOD trajectories.** We test whether the architectural transfer advantage depends on the two auxiliary losses ($\mathcal{L}_{\text{DIAYN}}$ and $\mathcal{L}_{\text{exc}}$) by training two ablated configurations: one with $\beta_{\text{DIAYN}} = 0$ (No DIAYN), and one with both $\beta_{\text{DIAYN}} = 0$ and $\lambda_{\text{exc}} = 0$ (No DIAYN, no exc). Both ablations retain the adapter architecture, phase mask, and task-identity objective. Final transfer trajectories on the two-petal (top) and butterfly (bottom) OOD tasks across the target and three configurations. The full configuration achieves numerically the best transfer hand L_1 on both OOD trajectories (two-petal: 0.005; butterfly: 0.011), and all three configurations achieve transfer well below the teacher baseline (two-petal: 0.04, butterfly: 0.058). The transfer advantage persists without DIAYN and excitation matching; task-identity supervision remains in all three conditions.

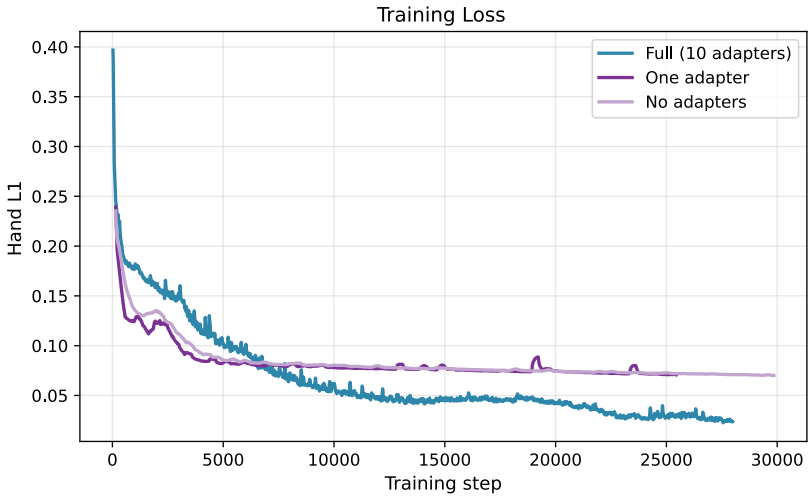

Figure 9: **Adapter bank ablation.** The previous ablation removed auxiliary supervision components but preserved the adapter architecture. We additionally test whether the adapter bank itself is crucial for the network's ability to learn the multi-task suite by training two further ablations: one with a single shared adapter (K=1, with all tasks routed through the same modulator), and one with no adapters at all (the cortex must learn all task dynamics within a single shared recurrence). For the latter, we allowed the decoder loss to train the LSTM recurrence.

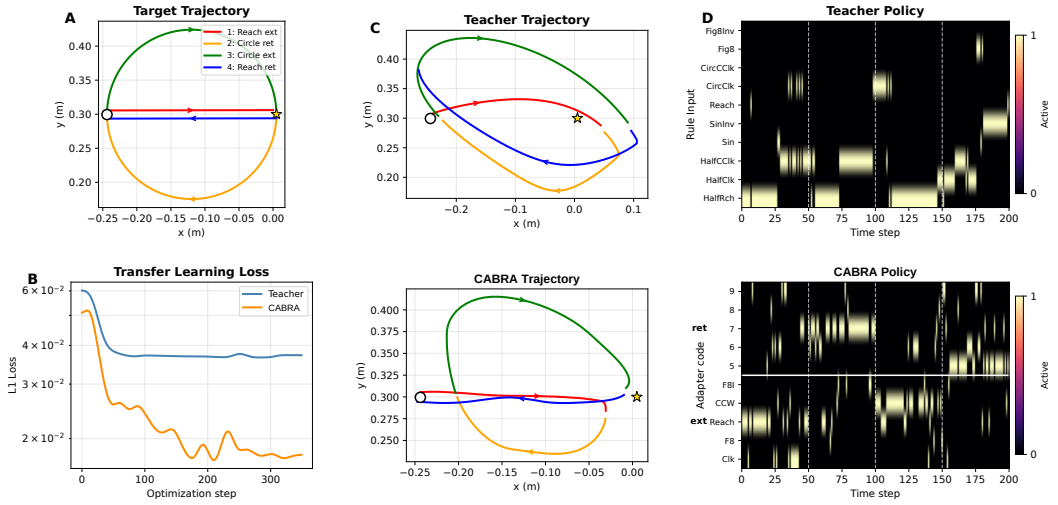

Figure 10: **Compositional transfer under hard discrete policy execution.** A complementary version of the two-petal transfer experiment in Figure 6, in which both networks are constrained to use *hard one-hot* selection over their respective task interfaces at every timestep. Optimization uses a straight-through estimator: the forward pass takes the argmax of the policy logits at each timestep (genuinely discrete selection), while gradients flow through the soft softmax distribution during backpropagation. **(A)** The two-petal target trajectory (same as Figure 6). **(B)** Smoothed L1 loss over 350 optimization steps. Both networks are optimized with identical Adam settings, target trajectory, and initialization heuristic; they differ only in the frozen substrate. Curves show median-filtered loss (window 25) followed by Gaussian smoothing ($\sigma = 8$) to suppress the noise inherent to straight-through estimation. Final mean L1 over the last 50 steps: CABRA 0.018, teacher 0.040, a $2.2\times$ gap; best L1: CABRA 0.014, teacher 0.036, a $2.5\times$ gap. **(C)** Final teacher (top) and CABRA (bottom) hand trajectories. Even under hard discrete selection, CABRA recovers the qualitative two-petal structure (orange and green half-loops), while the teacher's trajectory is contracted and distorted. **(D)** Final policies as actually applied to each network: the argmax code at each timestep. Top: teacher rule-input selection; bottom: CABRA adapter-code selection.

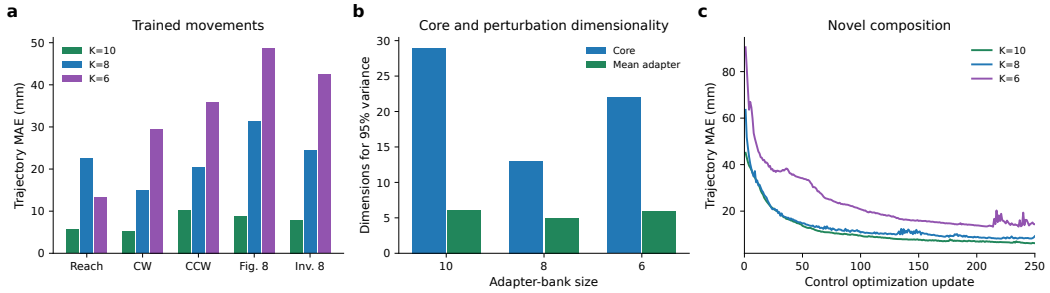

Figure 11: **Smaller adapter banks retain low-dimensional modulation and compositional transfer.** **(a)** Bars show mean trajectory error on the five trained movements for each adapter-bank size, evaluated on 32 demonstrations per movement. **(b)** Dimensionality of core activity and mean individual adapter perturbations, measured by the number of dimensions explaining 95% of variance using the core-focused rank protocol. **(c)** Optimization of new control programs for the two-petal trajectory over frozen networks with 10, 8 or 6 adapters, shown through update 250. Smaller banks preserve the ability to compose a new movement, with reduced accuracy in these runs.

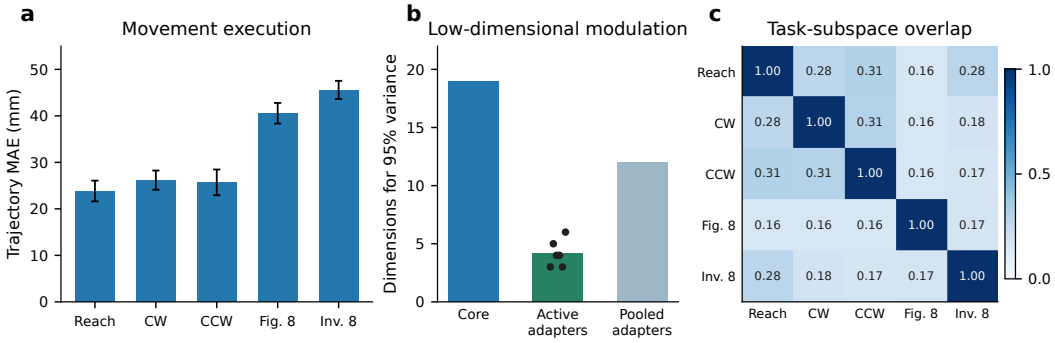

Figure 12: **Low-dimensional adapter perturbations and separated task representations emerge without a phase-split mask.** All ten adapters were available during both extension and retraction, retaining two inferred segments per movement. **(a)** Trajectory MAE on trained movements, averaged across time and both spatial coordinates; bars and error bars show mean $\pm$ SEM over 32 demonstrations per movement. **(b)** Core activity occupies 19 dimensions, individual active-adapter perturbations occupy 4.17 dimensions on average, and pooled perturbations occupy 12 dimensions (95% variance criterion). Dots show the six adapters engaged by the analysis programs, and the green bar shows their mean. The other four adapters were not recruited in these movements and thus left out. **(c)** Pairwise overlap of post-adapter task subspaces, measured as the mean squared cosine of principal angles using 25 principal components. Mean off-diagonal overlap is 0.219. Geometry was measured from closed-loop execution of the dominant inferred adapter pair for each task, with a midpoint transition.

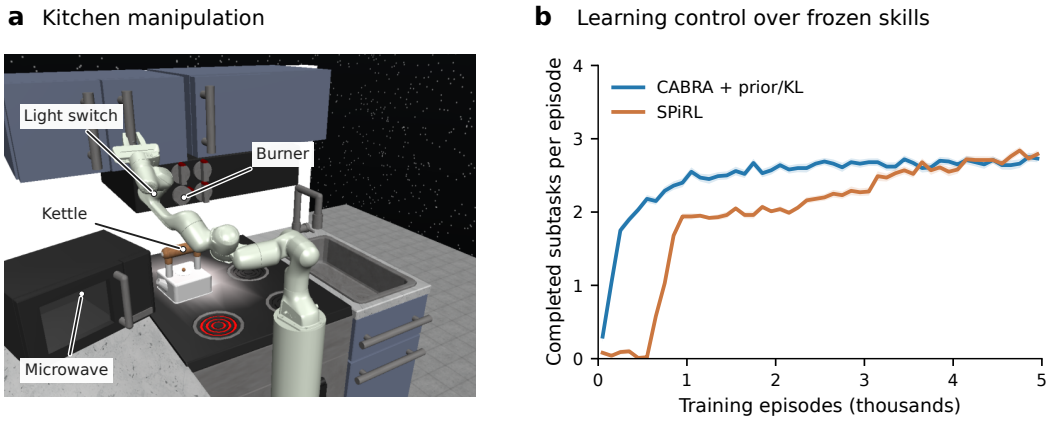

Figure 13: **CABRA supports efficient reinforcement learning in Kitchen. (a)** A simulated 3D robotic arm manipulates a microwave, kettle, burner and light switch. **(b)** After demonstration pretraining, a high-level policy learns to select skills over a frozen low-level controller. Both methods use a demonstration-derived skill prior and KL-regularized high-level reinforcement learning; CABRA uses a discrete adapter bank in place of SPiRL's latent skill interface. CABRA pretraining here uses no task-identity supervision. Curves show the mean number of completed subtasks in nonoverlapping blocks of 100 training episodes, plotted at block centers, for one run per method. Shading shows ± 1 SEM within each 100-episode block. CABRA acquires rewarded behavior earlier and reaches similar late performance. Mean reward over last 200 episodes is 2.74 for CABRA and 2.76 for SPiRL (maximum: four subtasks).

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 state three claims, each of which is empirically demonstrated in the corresponding results section: (i) the architecture discovers emergent low-rank perturbations despite no rank constraint (Figure 4, Section 4); (ii) task representations occupy highly separated subspaces while sharing a common dynamical substrate (Figure 5); (iii) a frozen network supports compositional transfer to novel out-of-distribution trajectories, outperforming a task-input-conditioned multitask baseline by an order of magnitude (Figure 6).

   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 section explicitly addresses limitations: the architecture uses task-identity supervision on the encoder's segment-1 representation, which shapes how codes are organized; a purely unsupervised version learning a variable number of codes and boundaries from task demands alone may yield qualitatively different organization. We also note that identifying the training paradigm that produces the most brain-like adapter library with minimal supervision remains a central open question.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper does not include theoretical results. Section 2 reviews the framework of Logiaco et al. [9], but no new theorems or proofs are claimed.

   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: Appendix A provides full architectural details, training procedure (loss composition, hyperparameters, optimizer settings, iteration counts), task and data details (biomechanical environment, compound task definitions, trial structure), and analysis methodology (effective rank, principal-subspace overlap, DSA, transfer optimization). The biomechanical simulator (`motornet`) is open-source and cited.

   Guidelines:

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

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: Training code for model attached to submission as zip file.

   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: Appendix A.2 specifies the optimizer and hyperparameters. Appendix A.3 specifies the task suite, trial structure, and reach conditions. Appendix A.4 specifies all analysis hyperparameters.

   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: Figure 4B reports bootstrap standard errors over $n = 10$ resamples for effective rank measurements.

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: All training and analysis experiments were run on a single GPU on an academic compute cluster, details mentioned in the appendix.

   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: The research uses simulated biomechanical control with no human or animal subjects, no scraped data, and no deployed systems. The work conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This is foundational research on neural network architectures for compositional motor control. The work has no direct path to deployment and no immediate societal impact concerns. It may indirectly contribute to neuroscience understanding of motor circuits and to the design of more flexible robotic controllers, both of which are widely viewed as positive directions.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release pretrained models, generative models, or scraped datasets. The trained motor controllers do not pose a risk of misuse.

    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: All external assets used in this work are properly cited: `motornet` [24] for biomechanical simulation, the multitask RNN framework of Lazzari and Saxena [10] as the teacher network, the CompILE framework of Kipf et al. [6] for the segmentation backbone, DIAYN [5] for the discriminator, and the DSA package of Ostrow et al. [16] for dynamical similarity analysis.

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: The paper does not release new assets at submission time. Code and trained models will be released upon publication, with documentation following the architectural and training details provided in Appendix A.

Guidelines:

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

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

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

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

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

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects, so IRB approval is not applicable.

    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: LLMs were not used as a component of the core methods. Any use was limited to code writing and paper writing assistance and does not impact the methodology, scientific rigor, or originality of the research.

    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.