APPENDICES

A Pullback and Combination of policies defined in subtask spaces

The priority metric and acceleration policy pair, $[\mathbf{M}_k, \pi_k]$, defined in subtask space $\mathbf{x} = \phi_k(\mathbf{q})$, can be pulled back to the root coordinates (configuration space for a robot) as follows. First, pre-multiplying the geometric policy through by the priority metric produces, $\mathbf{M}_k(\mathbf{x}, \dot{\mathbf{x}})\ddot{\mathbf{x}} + \mathbf{M}_k(\mathbf{x}, \dot{\mathbf{x}})\pi_k(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$, or written as

$$\mathbf{M}_k(\mathbf{x}, \dot{\mathbf{x}})\ddot{\mathbf{x}} + \mathbf{f}_k(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}, \tag{1}$$

where we additionally drop function variables for notational simplicity. Moreover, we have $\dot{\mathbf{x}} = \mathbf{J}_k\dot{\mathbf{q}}$ and $\ddot{\mathbf{x}} = \mathbf{J}_k\ddot{\mathbf{q}} + \dot{\mathbf{J}}_k\dot{\mathbf{q}}$, where $\mathbf{J}_k = \partial_{\mathbf{q}}\phi_k$. Substituting the acceleration equality, pre-multiplying through by $\mathbf{J}_k^T$, and rearranging yields,

$$\Big(\mathbf{J}_k^T\mathbf{M}_k\mathbf{J}_k\Big)\ddot{\mathbf{q}} + \mathbf{J}_k^T\big(\mathbf{f}_k + \mathbf{M}_k\dot{\mathbf{J}}_k\dot{\mathbf{q}}\big) = \mathbf{0}. \tag{2}$$

where we can further define $\widetilde{\mathbf{M}}_k = \Big(\mathbf{J}_k^T\mathbf{M}_k\mathbf{J}_k\Big)$ and $\widetilde{\mathbf{f}}_k = \mathbf{J}_k^T\big(\mathbf{f}_k + \mathbf{M}_k\dot{\mathbf{J}}_k\dot{\mathbf{q}}\big)$, resulting in $\widetilde{\mathbf{M}}_k\ddot{\mathbf{q}} + \widetilde{\mathbf{f}}_k = \mathbf{0}$. Multiple priority metric and geometry pairs in different spaces can then be combined together once they are pulled back as

$$\sum_k \widetilde{\mathbf{M}}_k\ddot{\mathbf{q}} + \sum_k \widetilde{\mathbf{f}}_k = \mathbf{0} \tag{3}$$

with a resulting system acceleration of $\ddot{\mathbf{q}} = -(\sum_k \widetilde{\mathbf{M}}_k)^{-1}\sum_k \widetilde{\mathbf{f}}_k$. This acceleration can then be forward integrated to obtain evolving the desired position and velocity states.

B Network Parameterization for Neural Geometric Fabrics

We parameterize all learnable components as multi-layer neural networks. For each geometry and priority metric pair, we separately parameterize the geometry $\{\pi_k\}_{k=1,2}$ and the priority metric $\{\mathbf{M}_k\}_{k=1,2}$ as a function of states and features. The geometries can be constructed with functions $\mathbf{h}_k(\mathbf{x}_k|\mathbf{p})$ scaled by $\|\dot{\mathbf{x}}_k\|^2$ to ensure the HD2 property, namely $\pi_k(\mathbf{x}_k, \dot{\mathbf{x}}_k|\mathbf{p}) = \mathbf{h}_k(\mathbf{x}_k|\mathbf{p})\|\dot{\mathbf{x}}_k\|^2$ for $k = 1, 2$, where $(\mathbf{x}_k, \dot{\mathbf{x}}_k)$ is the state represented in the space defined with mapping ϕ_k and $\mathbf{p}$ is the feature (e.g. the object pose). To ensure that the output of the priority metric network $\mathbf{M}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k|\mathbf{p})$ is positive definite, we follow the architecture described in [28]: a fully-connected neural network with output dimension $d(d+1)/2$ is used to predict the entries of a lower-triangular matrix $\mathbf{L}_k$, which serves as the Cholesky decomposition of the priority metric $\mathbf{M}_k$. Further, a small positive offset of 10^{-5} is added to the diagonal entries of $\mathbf{L}_k$ to ensure that the diagonal elements are strictly positive. Both $\mathbf{h}_k$ and $\mathbf{M}_k$ are represented by fully-connected neural networks with ReLU activation and two hidden layers with hidden units listed in Table 2.

For parameterization simplicity, the acceleration-based potential policy shares the same networks of the stacked space geometry, where $\mathbf{M}_f = \mathbf{M}_1$ and $\pi_f = \partial_{\mathbf{x}_1}\|\mathbf{h}_1(\mathbf{x}_1|\mathbf{p})\|$. The damping scalar $b = b(\mathbf{q}, \dot{\mathbf{q}}\|\mathbf{p})$ is represented by fully-connected neural networks with ReLU activation and two hidden layers with hidden units listed in Table 2, and can only produce non-negative values.

C Motion Generation with Neural Geometric Fabrics

Algorithm 1 and Algorithm 2 show how to generate motion with the learned NGF policy. These desired states are then tracked by a torque-level PD controller that is additionally gravity compensated.

Algorithm 1: NGF

input : state $(\mathbf{q}, \dot{\mathbf{q}})$ and feature $\mathbf{p}$.
output: an desired acceleration $\ddot{\mathbf{q}}_d$

for $k \leftarrow 0$ **to** K **do**
  $\mathbf{x}_k = \phi_k(\mathbf{q}), \dot{\mathbf{x}}_k = \mathbf{J}_k(\mathbf{q})\dot{\mathbf{q}}$ (push-forward);
  $\pi_k = \mathbf{h}_k(\mathbf{x}_k|\mathbf{p})\|\dot{\mathbf{x}}\|^2$ (geometry);
  $\mathbf{M}_k = \mathbf{M}_k(\mathbf{x}_k, \dot{\mathbf{x}}_k|\mathbf{p})$ (priority metric);
  $\ddot{\mathbf{q}}_k = \widetilde{\mathbf{M}}_k^{-1}\widetilde{\mathbf{f}}_k$ where $\widetilde{\mathbf{M}}_k = \mathbf{J}_k^T\mathbf{M}_k\mathbf{J}_k$, $\widetilde{\mathbf{f}}_k = \mathbf{J}_k^T\mathbf{M}_k\big(\pi_k + \dot{\mathbf{J}}_k\dot{\mathbf{q}}\big)$ (pull-back);
end
$\widetilde{\mathbf{M}}_f = \widetilde{\mathbf{M}}_K, \widetilde{\mathbf{f}}_f = \mathbf{J}_K^T\mathbf{M}_K\big(\partial_{\mathbf{x}_K}\|\mathbf{h}_K(\mathbf{x}_K|\mathbf{p})\| + \dot{\mathbf{J}}_K\dot{\mathbf{q}}\big)$ (potential policy);
$\ddot{\mathbf{q}}_d = -\mathbf{P}_e\big[\widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{f}}\big] - \widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{f}}_f - b\dot{\mathbf{q}}$ where $\mathbf{P}_e = (\mathbf{I} - \hat{\dot{\mathbf{q}}}\hat{\dot{\mathbf{q}}}^T)$, $\widetilde{\mathbf{M}} = \sum_k \widetilde{\mathbf{M}}_k + \widetilde{\mathbf{M}}_f$ and $b = b(\mathbf{q}, \dot{\mathbf{q}}|\mathbf{p}) \in \mathbb{R}^+$ (complete form);

Algorithm 2: Motion Generation with NGF

input : initial state $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$, feature $\mathbf{p}$, time step δt and task horizon T.
output: a desired state trajectory $\{(\mathbf{q}_0, \dot{\mathbf{q}}_0), (\mathbf{q}_1, \dot{\mathbf{q}}_1), ..., (\mathbf{q}_T, \dot{\mathbf{q}}_T)\}$ (we drop subscript d for notation simplicity.)

for $i \leftarrow 0$ **to** T **do**
  $\ddot{\mathbf{q}}_i = NGF(\mathbf{q}_i, \dot{\mathbf{q}}_i, \mathbf{p})$;
  $\mathbf{q}_{i+1} = \mathbf{q}_i + \dot{\mathbf{q}}_i\delta_t$;
  $\dot{\mathbf{q}}_{i+1} = \dot{\mathbf{q}}_i + \ddot{\mathbf{q}}_i\delta_t$;
end

D Policy Optimization

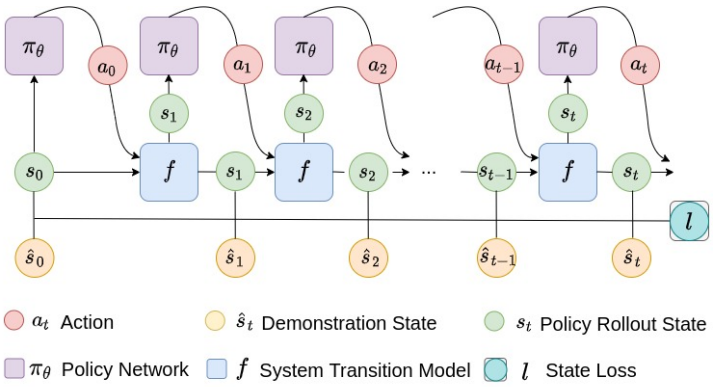


Figure 5: The architecture for policy training. π_θ is the policy network, which takes the current state s_t as input, and outputs the action a_t. f is the system transition model, which takes the state s_t and the action a_t as inputs, and outputs the next states s_{t+1}. $\hat{s}_t$ is the state in the demonstration, and $s_0 = \hat{s}_0$.

To mitigate the distribution shift issue involved in behavior cloning, we formulate the policy learning problem as a multi-step prediction error optimization problem as shown in Fig. 5, in which we learn a policy that can reproduce the demonstrated behavior through directly optimizing for the deviation from the trajectory demonstration $\hat{s}$ and the rollout trajectory s of the learned policy under the system transition function f:

$$\min_{\theta, \{s_t^{(i)}\}} \quad \frac{1}{2\sum_{i=1}^{N} T_i} \sum_{i=1}^{N} \sum_{t=0}^{T_i} \ell\left(\hat{s}_t^{(i)}, s_t^{(i)}\right) \tag{4}$$

$$\text{s.t.} \quad s_{t+1}^{(i)} = f\Big(s_t^{(i)}, \pi_\theta\big(s_t^{(i)}\big)\Big), \quad \forall i \in \{1, \dots, N\}, \quad t \in \{0, \dots, T_i - 1\}, \tag{5}$$

$$s_0^{(i)} = \hat{s}_0^{(i)}, \quad \forall i \in \{1, \dots, N\}. \tag{6}$$

where $\ell(\hat{s}, s) = \|\hat{s} - s\|_2^2$ measures state deviation. The constraints (5)–(6) ensure that $\{s_t^{(i)}\}$ is the rollout trajectory of policy π_θ under the transition function f initialized at $\hat{s}_0^{(i)}$. Since we use Geometric Fabrics as an acceleration-based policy, we consider a discrete-time acceleration-driven system, and the transition function f is defined as:

$$s_{t+1} = \begin{bmatrix} \mathbf{q}_{t+1} \\ \dot{\mathbf{q}}_{t+1} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_t + \dot{\mathbf{q}}_t \delta t \\ \dot{\mathbf{q}}_t + \pi_\theta(\mathbf{q}_t, \dot{\mathbf{q}}_t)\delta t \end{bmatrix} := f(s_t, \pi_\theta(s_t)) \tag{7}$$

where the state s_t is a tuple of position $\mathbf{q}_t \in \mathbb{R}^d$ and velocity $\dot{\mathbf{q}}_t \in \mathbb{R}^d$, the policy $\pi_\theta(\mathbf{q}_t, \dot{\mathbf{q}}_t)$ outputs the acceleration action, and δt is the sampling time. We then solve problem (4)–(6) using back-propagation-through-time [51, 52] with the Adam optimizer [53], where the learning rate is 10^{-2} and weight decay rate is 10^{-6}.

Note, for policies that generate velocity (or position offset) actions, we can simply replace the system transition function f with

$$s_{t+1} = s_t + \dot{s}_t \delta t := f(s_t, \pi_\theta(s_t)) \tag{8}$$

where the state s_t is the position and the policy $\pi_\theta(s_t)$ produces the velocity action, and δt is the sampling time.

The specific loss function for all policies is defined as:

$$\ell\left(\hat{s}_t^{(i)}, s_t^{(i)}\right) = \lambda \|\hat{\mathbf{q}}_t^{(i)} - \mathbf{q}_t^{(i)}\|_2^2 + \|\texttt{fk}(\hat{\mathbf{q}}_t^{(i)}) - \texttt{fk}(\mathbf{q}_t^{(i)})\|_2^2 \tag{9}$$

$$\text{with} \quad \mathbf{q}_0^{(i)} = \hat{\mathbf{q}}_0^{(i)}, \quad \forall i \in \{1, \dots, N\} \tag{10}$$

where `fk` is the forward kinematics mapping that maps the joint positions $\mathbf{q}$ to a vector that consists of the palm position and fingertips positions. N is the number demonstrations, T_i is the task horizon (number of steps) of the ith demonstration, and $\lambda = 0.1$ is a hyper-parameter that trades-off the losses of joint positions and the palm and fingertips positions. Hyper-parameters for training are listed in Table 3.

E Baseline Policies

We establish a variety of baseline policies with a spectrum of structure and action space. The intent is to systematically understand the effects of structure and action space on resulting performance and how the resulting performance compares to the proposed NGF. Some of the baselines follow state-of-the-art constructions or architectures while others serve as edifying reference points. All baseline policies were trained with the same loss and optimization settings as outlined in Appendix D. Neural network parameterization for each baseline is listed in Table 2.

E.1 Neural Network Acceleration Policy (NN)

The first baseline is a feedforward network that ingests 23-D joint positions and velocities, initial pose of the object, and target pose of the end-effector, and produces a 23 dimensional acceleration action. This action can then be time integrated to produce desired position and velocity signals. The significance of this baseline is to create a reference point of performance for a policy that produces a high-dimensional *acceleration* action without any additional structure. Acceleration actions are known to have a significant inductive bias in policy learning and dynamics modeling [31].

E.2 Riemannian Motion Policy (RMP)

The second baseline is an RMP which also takes as input state positions and velocities, initial pose of the object, and target pose of the end-effector, and produces a 23 dimensional acceleration action. The input states are calculated based on the leveraged maps as discussed in Section 4.1 and the 23-D joint position and velocity vectors. RMPs are a natural baseline for Geometric Fabrics because they are also second order systems and encapsulate Geometric Fabrics. We construct the RMP similar to the NGF policy, which is composed of policies defined in configuration space, and a stacked space of palm and hand Eigenspace. Policies in different spaces are created, pulled-back, and combined in the same way as Geometric Fabrics. However, the significance of this policy is to study the effect of including different task spaces in the policy structure without enforcing geometric policy structures or stability induced by potential and damping. The main difference here is that the priority metrics and acceleration policies in these spaces are directly generated by feedforward networks without imposing that the acceleration policy is a geometry. Leveraging the same spaces, pullback, and combination operations, the resulting RMP policy is then

$$\ddot{\mathbf{q}} = -\left(\widetilde{\mathbf{M}}_1 + \widetilde{\mathbf{M}}_2\right)^{-1} (\widetilde{\mathbf{f}}_1 + \widetilde{\mathbf{f}}_2). \tag{11}$$

E.3 Long Short-Term Memory Velocity Policy (LSTM)

We created an LSTM-based policy that takes as input 23 joint positions and the initial object pose and outputs a 23-dimensional configuration space position offset. Given an initial configuration, these prediction offsets are continuously summed, resulting in the full configuration space position trajectory. LSTMs are known to be beneficial to imitation learning [2] and can be useful for sequential processing and prediction architectures [56] (in this case, trajectory generation). This baseline does not leverage the task-relevant structures as with the NGF and RMP policies, but rather, its native recurrent structure is more directly studied to highlight how existing neural architectures perform in this problem space.

E.4 End-effector Neural Network Policy (EEF-GF)

The EEF-GF policy takes as input the previous and current end-effector pose, the previous and current finger joint positions, the initial object pose, and the target end-effector pose and produces actions in two different spaces: 1) palm pose offsets, and 2) hand joint position offsets. Given the initial conditions on palm pose and hand joints, these offsets are summed up over subsequent predictions, generating palm pose and hand joint angle trajectories. The palm pose trajectory is then given to an underlying hand-derived Geometric Fabric policy. This Geometric Fabric policy generates smooth joint space motion while converging to the target palm pose targets. Moreover, this policy resolves manipulator redundancy, postures the arm, avoids joint limits and joint speed limits, and avoids self-collision. In fact, this Geometric Fabric policy is exactly the same in design

and tuning as the one used in [29]. This EEF-GF policy serves to follow prior work in training policies with higher-level action spaces enabled by underlying hand-derived controllers [2, 57, 58], which have shown to be much more effective in imitation learning settings.

F Experimental Setup

We use DexPilot [50] to collect N demonstrations of a human operator teleoperating a robot consisting of a KUKA LBR iiwa 7 R800 robot arm and a Wonik Robotics Allegro robotic hand to perform all three tasks with objects placed in a variety of different poses. $N = 80$ for the first two tasks, and $N = 6$ for the third task. In total, the robot possesses 23 actuators. DexPilot functions by observing human hand motion and generating joint position commands for the entire robot. These commands are consumed by an underlying gravity-compensated PD controller that generates a desired drive torque for each of the actuators. During demonstration, the robot's commanded and measured joint position trajectories along with RGB-D camera data were recorded. After data collection, the commanded position trajectories were extracted. For the first two tasks, CosyPose [54] was used to label the initial camera image with the 6D pose of the object, and for third task, the orientation of the object was controlled via its placement on a printed template indicating orientation angles. The commanded trajectories along with this initial object pose reading were used in training the subsequent policies via imitation learning. Importantly, the policies are trained to reproduce these *commanded* joint position trajectories because following these trajectories with an underlying PD controller not only induces robot motion in freespace, but also generates contact forces when these commanded positions are kinematically infeasible and induce object-robot collision.

In general, it takes about 4 hours to collect 80 demonstrations for each task, and it takes about 3 hours to train the NGF policy on a single thread of a 4.0GHz Intel Core i7 CPU. This allows us to imbue complex robots with new manipulation skills within a day.

G Extended Experiment Discussion

The **imitation error** Δ is defined as:

$$\Delta = \sqrt{\frac{1}{\sum_{i=1}^{N} T_i} \sum_{i=1}^{N} \sum_{t=0}^{T_i} \|\texttt{fk}(\hat{\mathbf{q}}_t^{(i)}) - \texttt{fk}(\mathbf{q}_t^{(i)})\|_2^2} \quad \text{with} \quad \mathbf{q}_0^{(i)} = \hat{\mathbf{q}}_0^{(i)}, \quad \forall i \in \{1, \ldots, N\} \tag{12}$$

where $\texttt{fk}$ is the forward kinematics mapping that maps the joint positions $\mathbf{q}$ to a vector that consists of the palm position and fingertips positions. N is the number demonstrations and T_i is the task horizon (number of steps) of the ith demonstration. The imitation error Δ is an indicator of policy performance and generalizability.

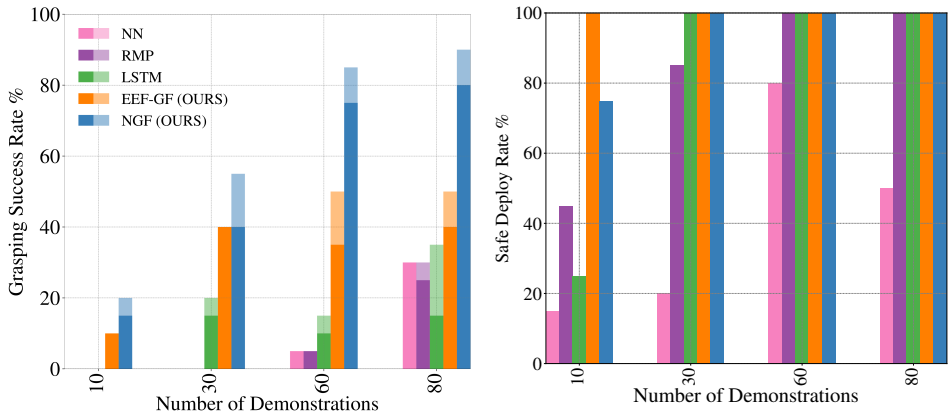
G.1 Analysis on Sample Efficiency

There are a number of conclusions that can be drawn when studying the imitation error and task performance across all policies and number of demonstrations (see Fig. 4). First, as one would expect, all policies benefit from increasing amounts of training demonstrations as both the imitation error and real-world performance inversely correlate to the number of demonstrations. Second, imitation error correlates with the real-world performance, although this is not a perfect one-to-one correspondence (see all three plots in Fig. 4). More specifically, the NN policy which only possesses an inductive bias via its acceleration action consistently has the highest imitation error which translated to the lowest combination of task success rates and safe deploy rates. The NGF has the lowest mean imitation error (and smallest variance) across all quantities of training demonstrations which translated to the best combination of task success rates and safe deploy rates. The RMP had the second largest imitation error, which translated to poor task success rates, but better safety deploy rates relative to the NN policy. This indicates that the additional task structure of the RMP was indeed beneficial. Third, the small difference in imitation error between the LSTM and EEF-GF policies did not quite translate in the same way to the real world performance. Although, these two policies were the second and third highest performing, matching their imitation error positioning relative to the other policies. The EEF-GF policy has the second highest level of performance, which matches observations in the literature that policies trained in the end-effector space can improve performance.

Overall, the NGF's impressive performance must be attributed to its additional structure imposed by Geometric Fabrics: 1) encoding policies primarily as geometries, which are speed-invariant, and 2) being a stable dynamical system with convergence offered by the potential policy and damping. This rich structure is well studied in both [30] and [29], and allows us to learn provably stable second order policies that extend beyond classical mechanics.

G.2 Sugar box task

Here we present the full result on the task performance of each policy trained on demonstration trajectories of sizes $[10, 30, 60, 80]$, deployed on the real robot and evaluated on 20 arbitrary object poses with two attempts. If the first deployment did not succeed, a second attempt is initiated without moving the object, offering the policy a second chance to grasp the object. Task success rates and safe deployment rate were calculated for each policy and training dataset size as shown in Section G.2, where the solid color bars indicate rates after the first attempt and faded color bars indicate rates after the second attempt.



(a) (b)

Figure 6: Policy performance based on (a) percentage of deployments that resulted in successfully lifting the object (darker bars indicate success rate of first attempt, faded bars indicate success rate after second attempt), and (b) percentage of deployments that were safe to deploy on the real robot.

Remark: In general, the grasp success rate does not fully capture failure modes for policy performance. Although grasp performance for either EEF-GF and LSTM do not scale as well beyond 30 training demonstrations, we empirically observed improved grasping behaviors for policies trained with larger amounts of data. For instance, policies typically yielded better object approach and caging behaviors than ones trained with fewer demonstrations. However, policies often still fail the grasping exercise due to lack of fine finger placement and appropriate squeezing motions. For instance, if fingers miss the edge of a box by only a few millimeters, then the grasp will likely fail (see Fig. 7 for illustration). Indeed, successful grasping requires sufficiently precise and coordinated motion among all fingers.

An example result of the learned NGF is presented in Fig. 8.

G.3 Coffee can task

From Fig. 3, we observed that our approach resulted in a 100% deployment rate and significantly outperformed the baselines in terms of success rate, as it is 70% for the NGF and no more than 50% for all the baselines. An example result of the learned NGF is presented in Fig. 9.

G.4 Toolbox task

In this task, we test policy generalization to unseen toolbox orientations, including both in distribution and out of distribution cases. Testing results are presented in Table 1, where "S", "F", and

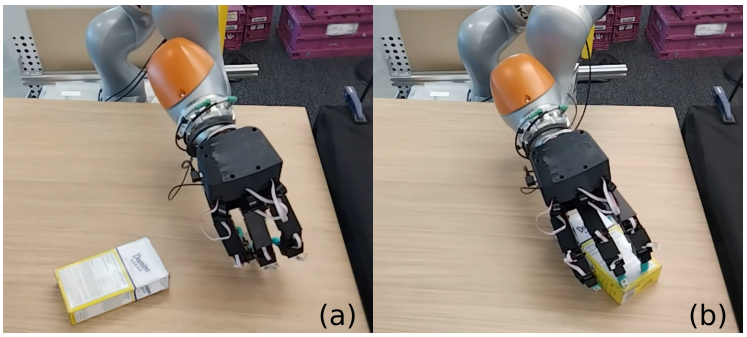


Figure 7: Typical EEF-GF policy grasp acquisition failures when trained on (a) 30 demonstrations, and (b) 80 demonstrations.

Figure 8: Learned NGF policy that intrinsically generates smooth motion across all 23 actuators of the robot to approach, grasp, and lift an object.

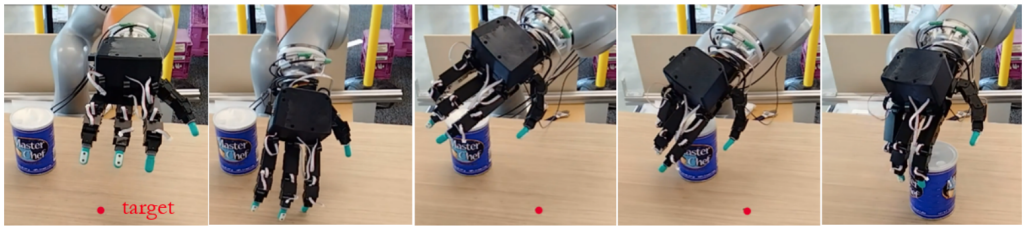


Figure 9: Learned NGF that intrinsically generates smooth motion across all 23 actuators of the robot to touch the target point while avoiding collision with the object and then push the object over to the target location.

"U" represent "Success", "Failure" and "unsafe to deploy", respectively. An example result of the learned NGF is presented in Fig. 10.

Table 1: Testing Results for Toolbox Task.

Results		Orientation (degree)								
		-10	0	20	40	60	80	100	120	130
Policy	NN	U	S	S	S	S	F	S	U	U
	RMP	U	S	F	S	S	S	S	S	F
	LSTM	U	S	S	S	S	S	S	U	U
	EEF-GF	S	S	S	S	F	F	S	S	F
	NGF	F	S	S	S	S	S	S	S	S

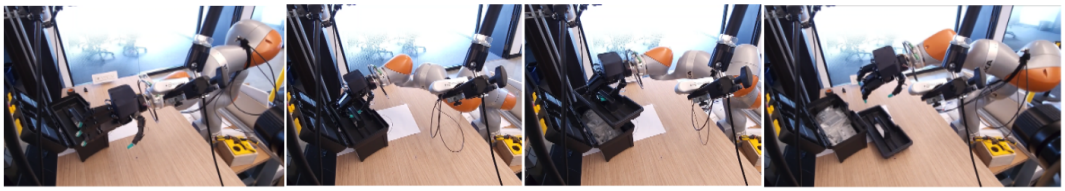


Figure 10: Learned NGF that intrinsically generates smooth motion across all 23 actuators of the robot to grasp the top tray of a toolbox and place it on the table.

H Reactivity of the Learned NGF Policy

To illustrate that NGFs are fundamentally reactive policies, we deployed the learned NGF policy on the coffee can pushing task as illustrated in Fig. 11. While the robot was traveling to grasp the coffee can, a person pushed the coffee can into a different pose. The NGF, which now ingests the real-time pose measurement of the coffee can, immediately and smoothly redirects the robot hand to the change in coffee can position and pushes it to the target destination to complete the task. We tried this in a few different places and observed that: 1) object pose detection can fail causing deployment issues (as we pointed out earlier), and 2) if the coffee can is pushed too far away from the starting position while the hand is too close to the can, then the robot can get confused and not complete the task. During the latter case, we never observed destructive motion induced by the NGF policy, which was reassuring. However, to obtain truly good performance in closed-loop control, we believe the NGF policy will need to be trained on a much larger dataset and allow the policy to observe shifting object pose during training at various time points in the trajectory, not just the beginning.

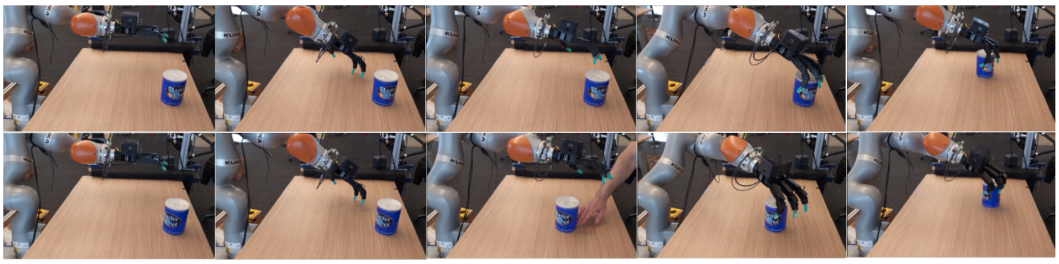


Figure 11: The learned NGF policy reacts to changes of the object location during execution in the coffee can task. First row shows behavior without moving the object. Second row shows behavior when the position of the object is changed while the robot is en route to grasping the object.

I Hyper-parameters for networks and training

Table 2 lists the network hyper-parameters used for each policy that we studied in this paper and they are selected through a set of ablation experiments as outlined in Appendix J. Table 3 lists the hyper-parameters used for the training.

Table 2: Hyper-parameters of Policy Networks

Policy names	Network	Hidden units for Task 1&2	Hidden units for Task 3	Activation
NGF	MLP	(128, 64)	(64, 32)	ReLU
RMP	MLP	(128, 64)	(64, 32)	ReLU
NN	MLP	(512, 256)	(256, 128)	ReLU
EEF-GF	MLP	(512, 256)	(256, 128)	ReLU
LSTM	LSTM	256	128	Tanh
	MLP	(128, 64)	None	ReLU

Table 3: Hyper-parameters for training

Hyperparamter names	Values
Training epochs	2000
Learning rate	0.001
Optimizer	ADAM
Decay	1e-6
Clip norm	1.

J Ablation study on hyper-parameters for network sizes

The hyper-parameters listed in Table 2 are selected through a set of ablation experiments on the imitation error as discussed in Section 6.3, where we studied the imitation error for each policy on the first task over three sets of network sizes, including small size, median size and large size as listed in Table 4. The imitation error for each policy is shown in Fig. 12. Based on these results, we decided to use the network size that gives the best performance on imitation error for each policy, namely large network for NN and EEF-GF, median network for LSTM, and to keep RMP and NGF similar to each other, we use the median network for them. Since there are only 6 demonstrations for task 3, we decide to use a smaller network size for each policy to prevent overfitting to the training data, namely median network for NN and EEF-GF, small network for LSTM, and small network for RMP and NGF.

Table 4: Hyper-parameters on Network Sizes

Policy names	Network	Small Size	Median Size	Large Size
NGF	MLP	(64, 32)	(128, 64)	(256, 128)
RMP	MLP	(64, 32)	(128, 64)	(256, 128)
NN	MLP	(128, 64)	(256, 128)	(512, 256)
EEF-GF	MLP	(128, 64)	(256, 128)	(512, 256)
LSTM	LSTM	128	256	512
	MLP	None	(128, 64)	(256, 128)

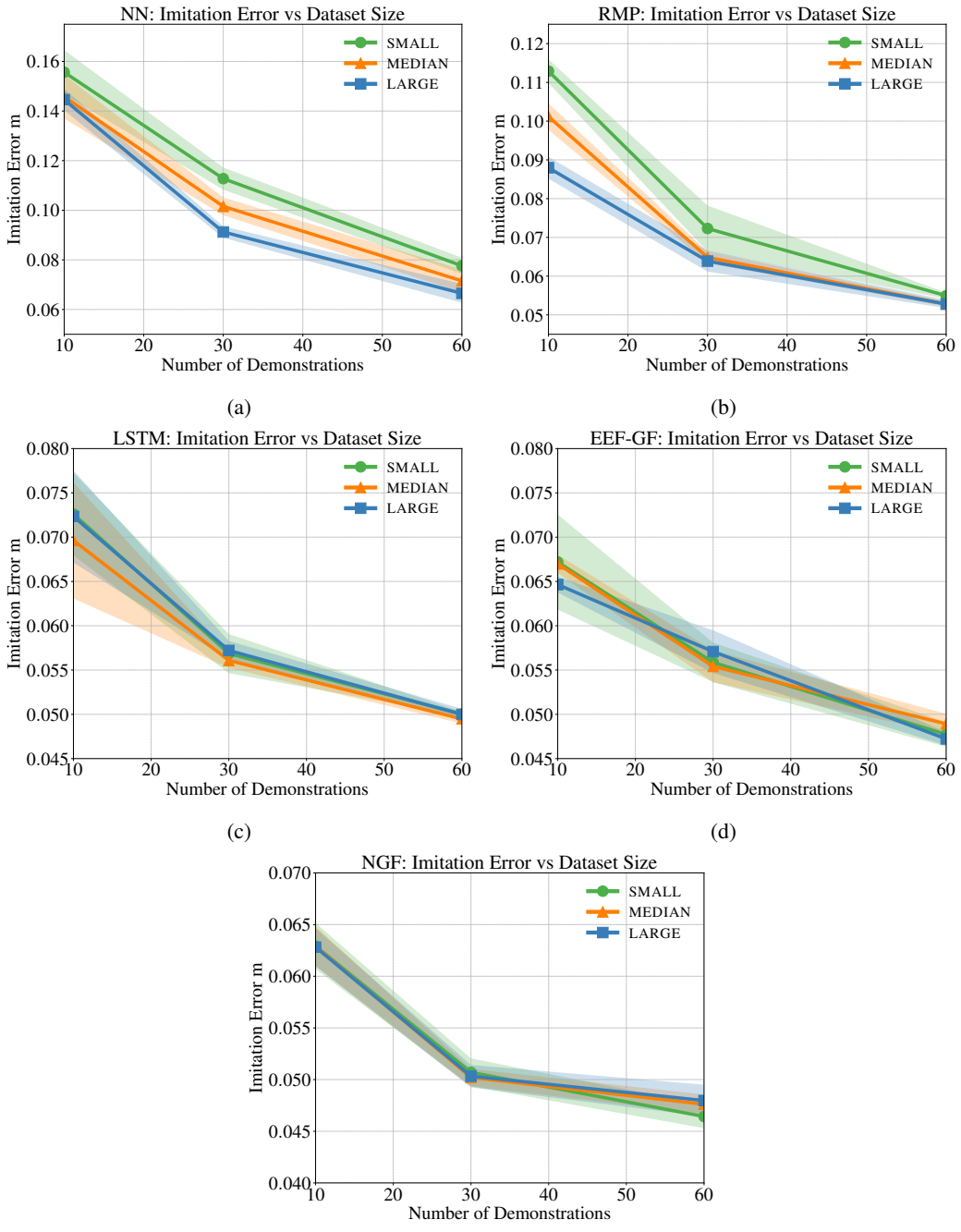
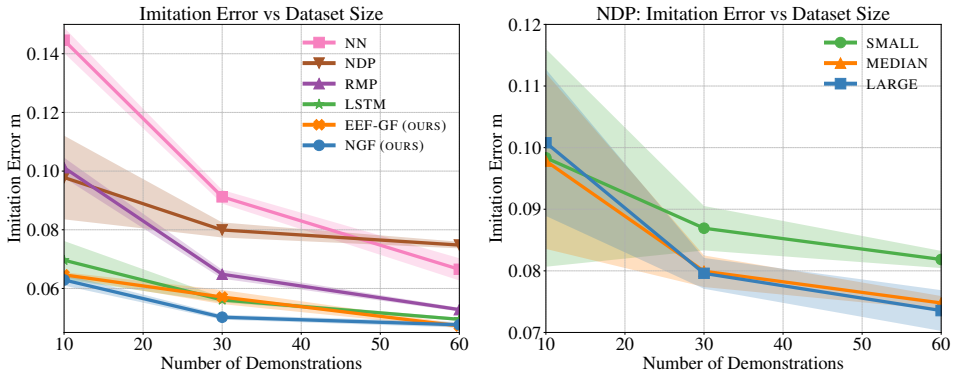


Figure 12: Imitation error for each policy on the first task over small, median and large network sizes for each policy: (a) NN, (b) RMP, (c) LSTM, (d) EEF-GF, (e) NGF.

K An additional baseline - Neural Dynamic Policy (NDP)

An NDP [59] parameterizes the second order Dynamics Motion Primitives (DMP) with neural networks, and can be utilized to learn behaviors from demonstrations. We train an NDP following the imitation learning formulation described in [59] with our loss function. The NDP evaluates its neural network components at the beginning of each task given the initial object pose and the target position to produce variables in the DMP, which is then forward integrated to produce the commanded trajectory given the initial joint positions and velocities.

This structure is good for avoiding distribution shift issues that other methods suffer from and alleviating the overfitting issue when there are only a few demonstrations. However, it diminishes its capacity in encoding the fine details of the behavior, which are very critical in dexterous manipulation with high dimensional systems. As shown in Fig. 13a, thanks to the strong inductive bias, NDP outperforms NN when trained with less than 30 demonstrations. However, due to the lack of expressivity, NDP is outperformed by NN when trained with 60 demonstrations. Overall, the NDP has the worst imitation error of all policies in the higher data regime and significantly worse imitation error than the LSTM, EEF-GF, and NGF policies in general. Given these results, we expect the NDP to have real world performance somewhere between the NN and RMP policies, neither of which can compete with the NGF policy in the real world experiments. We also ran ablation studies across different neural network sizes just like all our policies (see Fig. 13b) and decided on the median network size [256, 128] with 10 radial basis functions (RBFs) given its best performance in imitation error.



(a) (b)

Figure 13: (a) Imitation error of each policy, and (b) Imitation error for NDP on the first task over small network size [128, 64] with 5 RBFs, median network size [256, 128] with 10 RBFs and large network size [512, 256] with 20 RBFs. Solid lines indicate the mean and the shaded area show mean ± standard deviation, over the 5 random seeds.

L Comparison between NGFs and RMPs

Here we provide a more detailed comparison between NGF and RMP. The key similarities are 1) both NGFs and RMPs are dynamical-systems-based structured policies which can be learned from demonstrations; and 2) both NGFs and RMPs leverage exactly the same tree structure of subtask spaces. It is important to note that RMP does not necessarily represent a competing approach to NGF. In fact, Geometric Fabrics can be viewed as specialized versions of RMPs with more effective structural inductive biases. Indeed, our key contributions are to enable learning Geometric fabrics directly from demonstrations and to demonstrate that the inductive bias in our approach is considerably more data efficient than that introduced by RMP within the context of high-dimensional dexterous manipulation skills. To this end, we solve a number of challenges and our approach results in the following benefits over RMPs:

1. **Geometric fabrics provide a unique and beneficial inductive bias.** The combined works of [29] and [30] set the theoretical foundations culminating in second order systems that can be classified as a Geometric Fabric. A Geometric Fabric requires four main components: 1) a system geometry, 2) a Finsler energy that we will force the geometry to conserve, 3) a potential function, and 4) a strictly positive scalar acceleration damper. This combined machinery was only recently introduced in [29] and [30] and no prior work has demonstrated that this structured class of policies can be learned from demonstrations. Our experiments show that the coordinated high-dof dexterous manipulation demonstrations are well-modeled as geometries, which separate the motion (paths) from speed of traversal (an inductive bias that allows the policy to learn similar motions from trajectories that may have the same shape but different speed profiles).
2. **Leveraging the theory of geometric fabrics in a learning context requires nontrivial design not addressed in [29] and [30].** We introduce how to parameterize the NGF with neural networks while ensuring that the theoretical guarantees of Geometric Fabrics are maintained. Specifically, we construct two HD2 geometric policies, one potential policy, their associated priority metrics, and a damper via neural networks. We also choose to energize these geometries via a specific energy established in configuration space and establish a novel tree of spaces that these components reside within (more discussion below).
3. **Our training technique and design of loss functions represent steps beyond prior work.** These decisions were not investigated or considered in existing RMP training papers, and required significant iteration before we converged on the proposed learning algorithm. These choices greatly impacted performance.
4. **Our choice of action spaces (the configuration space and concatenated PCA space for the hand and 3D Euclidean space at the palm point) is nontrivial and is unique to our work.** We spent significant design cycles iterating on which task spaces performed best for both RMP and NGF policies. We believe that these insights will be highly relevant to future efforts aimed at learning dexterous manipulation policies, even if a completely different policy parametrization is utilized.
5. **The novelty of our work (or that of RMP) does not reside on task space trees.** In fact, trees of task spaces have a long history in the robotics literature dating from much earlier than RMPs or geometric fabrics. RMPs and especially RMPflow introduce a data structure that's helpful for leveraging pullback in solving structure task space least squares problems, and that's used in geometric fabrics and our work here (geometric fabrics actually add nontrivial machinery based on the theory in [29] and [30]). But the use of transform trees here is not our focus. Our work demonstrates that the inductive bias of geometric fabrics is pivotal to achieving data efficiencies low enough to make learning high-dof dexterous manipulation from human demonstrations on physical hardware possible.