# Appendices

## A KODex Pseudo-code

The overall pseudo-code for KODex is shown below.

**Algorithm 1:** KODex

```
Demonstration Data Collection
Initialize $D = \varnothing$;
for $n \in \{1, ..., N\}$ do
    Generate a $T^{(n)}$-horizon trajectory of states and torques $\{[\mathrm{x}^n(t), \tau^n(t)]\}_{t=1}^{t=T^{(n)}}$;
    Add $\{[\mathrm{x}^n(t), \tau^n(t)]\}_{t=1}^{t=T^{(n)}}$ to $D$;
end
Koopman Operator Approximation
Determine lifting function $\phi(\mathrm{x}(t))$;
Compute $\mathbf{K}$ on $D$ (6, 9);
Controller Design
Build a controller $C$ as a neural network with inputs as $(\mathrm{x}_r(t), \mathrm{x}_r(t+1))$ and output as $\tau(t)$;
Train $C$ using state-torque pairs $(\mathrm{x}_r^n(t), \mathrm{x}_r^n(t+1), \tau^n(t))$ in $D$ (10);
Implementation
Specify the initial states $\mathrm{x}(1)$;
for $t \in \{1, ..., T-1\}$ do
    Predict the next robot states $\hat{\mathrm{x}}_r(t+1)$ using $\mathbf{K}$ (3 8);
    Read the current robot states $\mathrm{x}_r(t)$;
    Generate the torque $\tau(t)$ using $C$ on $(\mathrm{x}_r(t), \hat{\mathrm{x}}_r(t+1))$ and execute it;
end
```

## B State Design

In this section, we show the state design for each task in detail. It should be noted that the motion capability of the hand for each task were suggested from the work [8] that originally introduced these tasks. For a decent implementation, we employed the same setting.

**Tool use** For this task, the floating wrist base can only rotate along the $x$ and $y$ axis, so we have $\mathrm{x}_r(t) \in \mathcal{X}_r \subset \mathbb{R}^{26}$. Regarding the object states, unlike the other tasks, where the objects of interest are directly manipulated by the hand, this task requires to modify the environment itself. As a result, except for the hammer positions, orientations and their corresponding velocities $\mathrm{p}_t^{\text{tool}}, \mathrm{o}_t^{\text{tool}}, \dot{\mathrm{p}}_t^{\text{tool}}, \dot{\mathrm{o}}_t^{\text{tool}}$ ($\mathbb{R}^3$), we also define the nail goal position $\mathrm{p}^{\text{nail}}$ ($\mathbb{R}^3$). Finally, we have $\mathrm{x}_o(t) = [\mathrm{p}_t^{\text{tool}}, \mathrm{o}_t^{\text{tool}}, \dot{\mathrm{p}}_t^{\text{tool}}, \dot{\mathrm{o}}_t^{\text{tool}}, \mathrm{p}^{\text{nail}}] \in \mathcal{X}_o \subset \mathbb{R}^{15}$. As a result, $\mathrm{x}(t)$ includes 41 states in total and we use $T = 100$.

**Door opening** For this task, the floating wrist base can only move along the direction that is perpendicular to the door plane but rotate freely, so we have $\mathrm{x}_r(t) \in \mathcal{X}_r \subset \mathbb{R}^{28}$. Regarding the object states, we define the fixed door position $\mathrm{p}^{\text{door}}$, which can provide with case-specific information (similar to $\mathrm{p}^{\text{nail}}$ in Tool Use), and the handle positions $\mathrm{p}_t^{\text{handle}}$ (both $\mathbb{R}^3$). In order to take into consideration the status of door being opened, we include the angular velocity of the opening angle $v_t(\mathbb{R}^1)$. Finally, we have $\mathrm{x}_o(t) = [\mathrm{p}_t^{\text{handle}}, v_t, \mathrm{p}^{\text{door}}] \in \mathcal{X}_o \subset \mathbb{R}^7$. As a result, $\mathrm{x}(t)$ includes 35 states in total and we use $T = 70$.

**Object relocation** For this task, the ADROIT hand is fully actuated, so we have $\mathrm{x}_r(t) \in \mathcal{X}^r \subset \mathbb{R}^{30}$ (24-DoF hand + 6-DoF floating wrist base). Regarding the object states, we define $\mathrm{p}^{\text{target}}$ and $\mathrm{p}_t^{\text{ball}}$ as the target and current positions. Then, we compute $\bar{\mathrm{p}}_t^{\text{ball}} = \mathrm{p}_t^{\text{ball}} - \mathrm{p}^{\text{target}}$, which is the component of $\mathrm{p}_t^{\text{ball}}$ in a new coordinate frame that is constructed by $\mathrm{p}^{\text{target}}$ being the origin. We additional include the ball orientation $\mathrm{o}_t^{\text{ball}}$ and their corresponding velocities $\dot{\mathrm{p}}_t^{\text{ball}}$, $\dot{\mathrm{o}}_t^{\text{ball}}$ (all $\mathbb{R}^3$). Finally, we have $\mathrm{x}_o(t) = [\bar{\mathrm{p}}_t^{\text{ball}}, \mathrm{o}_t^{\text{ball}}, \dot{\mathrm{p}}_t^{\text{ball}}, \dot{\mathrm{o}}_t^{\text{ball}}] \in \mathcal{X}_o \subset \mathbb{R}^{12}$. As a result, $\mathrm{x}(t)$ includes 42 states in total and we use

$T = 100$.

**In-hand reorientation** For this task, the floating wrist base is fixed, so we only consider the 24-DoF hand joints. Therefore, we have $\mathrm{x}_r(t) \in \mathcal{X}_r \subset \mathbb{R}^{24}$. Regarding the object states, we define $\mathrm{o}^{\text{goal}}$ and $\mathrm{o}_t^{\text{pen}}$ as the goal and current pen orientations, which are both unit direction vectors. Then, we transform $\mathrm{o}_t^{\text{pen}}$ to a new rotated coordinate frame that is constructed by $\mathrm{o}^{\text{goal}}$ being $x$ axis ([1,0,0]). Note that the vector $\bar{\mathrm{o}}_t^{\text{pen}}$ after transformation is also a unit vector and it converges to x axis if the pen is perfectly manipulated to goal orientation $\mathrm{o}^{\text{goal}}$. In addition, we also include the center of mass position $\mathrm{p}_t^{\text{pen}}$ and their corresponding velocities $\dot{\mathrm{p}}_t^{\text{pen}}$, $\dot{\mathrm{o}}_t^{\text{pen}}$ (all $\mathbb{R}^3$). Finally, we have $\mathrm{x}_o(t) = [\mathrm{p}_t^{\text{pen}}, \bar{\mathrm{o}}_t^{\text{pen}}, \dot{\mathrm{p}}_t^{\text{pen}}, \dot{\mathrm{o}}_t^{\text{pen}}] \in \mathcal{X}_o \subset \mathbb{R}^{12}$. As a result, $\mathrm{x}(t)$ includes 36 states in total and we use $T = 100$.

In this work, we only included the joint positions as the robot states (with the only exception of NGF's second-order policy) for the following reasons: 1) Given that these tasks are not repetitive, we found that joint position information was sufficient to disambiguate the robot's next action, 2) even when ambiguity arises for a given joint position, object state information can help with disambiguation. Further, the impressive performance achieved by KODex in our experiments support this design choice. Indeed, KODex is agnostic to this specific state design. One can incorporate velocity information into the robot state space without the need of any changes to the training procedure.

## C Task Success Criteria

The task success criteria are listed below. The settings were the same as proposed in [8].

**Tool Use:** The task is considered successful if at last time step $T$, the Euclidean distance between the final nail position and the goal nail position is smaller than 0.01.

**Door Opening:** The task is considered successful if at last time step $T$, the door opening angle is larger than 1.35 rad.

**Object Relocation:** At each time step $t$, if $\sqrt{|\mathrm{p}^{\text{target}} - \mathrm{p}_t^{\text{ball}}|^2} < 0.10$, then we have $\rho(t) = 1$. The task is considered successful if $\sum_{t=1}^{T} \rho(t) > 10$.

**In-hand Reorientation:** At each time step $t$, if $\mathrm{o}^{\text{goal}} \cdot \mathrm{o}_t^{\text{pen}} > 0.90$ ($\mathrm{o}^{\text{goal}} \cdot \mathrm{o}_t^{\text{pen}}$ measures orientation similarity), then we have $\rho(t) = 1$. The task is considered successful if $\sum_{t=1}^{T} \rho(t) > 10$.

## D Sampling Procedure

We describe the sampling procedure in this section. The sample distributions used for RL training and demo collection were identical, as suggested in [8]. The out-of-distribution data were generated to evaluate the zero-shot out-of-distribution generalizability of each policy.

**Tool Use:** We randomly sampled the nail heights ($h$) from a uniform distributions. Within distribution: we used $h \in \mathcal{H} \sim \mathcal{U}(0.1, 0.25)$; Out of distribution: we used $h \in \mathcal{H} \sim \mathcal{U}(0.05, 0.1) \cup \mathcal{U}(0.25, 0.3)$.

**Door Opening:** We randomly sampled the door positions ($xyz$) from uniform distributions. Within distribution: we used $x \in \mathcal{X} \sim \mathcal{U}(-0.3, 0)$, $y \in \mathcal{Y} \sim \mathcal{U}(0.2, 0.35)$, and $z \in \mathcal{Z} \sim \mathcal{U}(0.252, 0.402)$; Out of distribution: we used $y \in \mathcal{Y} \sim \mathcal{U}(0.15, 0.2) \cup \mathcal{U}(0.35, 0.4)$ ($x, z$ remained unchanged).

**Object Relocation:** We randomly sampled the target positions ($xyz$) from uniform distributions. Within distribution: we used $x \in \mathcal{X} \sim \mathcal{U}(-0.25, 0.25)$, $y \in \mathcal{Y} \sim \mathcal{U}(-0.25, 0.25)$, and $z \in \mathcal{Z} \sim \mathcal{U}(0.15, 0.35)$; Out of distribution: we used $z \in \mathcal{Z} \sim \mathcal{U}(0.35, 0.40)$ ($x, y$ remained unchanged).

**In-hand Reorientation:** We randomly sampled the pitch ($\alpha$) and yaw ($\beta$) angles of the goal orientation from uniform distributions. Within distribution: we used $\alpha \in \mathcal{A} \sim \mathcal{U}(-1, 1)$ and $\beta \in \mathcal{B} \sim \mathcal{U}(-1, 1)$; Out of distribution: we used $\{(\alpha, \beta) \in (\mathcal{A}, \mathcal{B}) \sim (\mathcal{U}(-1, 1.2)), \mathcal{U}(1, 1.2)) \cup (\mathcal{U}(1, 1.2)), \mathcal{U}(-1.2, 1)) \cup (\mathcal{U}(-1.2, 1)), \mathcal{U}(-1.2, -1)) \cup (\mathcal{U}(-1.2, -1)), \mathcal{U}(-1, 1.2))\}$.

# E Policy Design

We show the detailed policy design in this section. All the baseline policies were trained to minimize the trajectory reproduction error.

**KODex:** The representation of the system is given as: $\mathrm{x}_r = [x_r^1, x_r^2, \cdots, x_r^n]$ and $\mathrm{x}_o = [x_o^1, x_o^2, \cdots, x_o^m]$ and superscript is used to index states. The details of the state design for each task is provided in Appendices B. In experiments, the vector-valued lifting functions $\psi_r$ and $\psi_o$ in (8) were polynomial basis function defined as

$$\begin{aligned} \psi_r =& \{x_r^i x_r^j\} \cup \{(x_r^i)^3\} \text{ for } i, j = 1, \cdots, n \\ \psi_o =& \{x_o^i x_o^j\} \cup \{(x_o^i)^2 (x_o^j)\} \text{ for } i, j = 1, \cdots, m \end{aligned} \tag{11}$$

Note that $x_r^i x_r^j / x_r^j x_r^i$ only appears once in lifting functions (similar to $x_o^i x_o^j / x_o^j x_o^i$), and we ignore $t$ as the lifting functions are the same across the time horizon.

The choice of lifting functions can be viewed as the hyper-parameter of KODex. We make this choice as inspired from [24] and experimental results also indicate its effectiveness. Through all the experiments, we sticked with the same set of lifting functions, which helped to relieve us from extensive efforts of tuning the hyper-parameters, e.g. network layer size, that were necessary for baseline policies as shown in Appendices F.

**Full-connected Neural Network (NN):** The first baseline is a feedforward network that ingests the states $\mathrm{x}(1)$ and iteratively produces the predictions $\mathrm{x}(t), t = 2, \cdots, T$ via the rollout of a Multilayer Perceptron (MLP). The reference joint trajectories ($\mathrm{x}_r(t)$) are then used to execute the robot with the learned controller $C$. The significance of this baseline is to evaluate a policy that produces a high-dimensional motion without any additional structure.

**Long Short-Term Memory (LSTM):** We create an LSTM-based policy under the same input-output flow as the NN policy. We also apply two fully-connected layers between the task input/output and the input/hidden state of the LSTM network. Similarly, the same controller $C$ is deployed to track the reference joint trajectory. LSTM networks are known to be beneficial to imitation learning [30] and suitable for sequential processing [37], e.g, motion generation. Therefore, we expect to evaluate the performance of the recurrent structures in these tasks.

**Neural Dynamic Policy (NDP):** The Neural Dynamic Policy [17] embeds desired dynamical structure as a layer in neural networks. Specifically, the parameters of the second order Dynamics Motion Primitives (DMP) are predicted as outputs of the preceding layers (MLP in [17]). As a result, it allows the overall policy easily reason in the space of trajectories and can be utilized for learning from demonstration. We train an NDP policy following the imitation learning pipeline described in [17]. For each task, given $\mathrm{x}(1)$, the neural network components in NDP generate the parameters of DMPs (radial basis functions (RBFs) in [17]), which are forward integrated to produce the reference joint trajectories for tracking.

**Neural Geometric Fabrics policy (NGF):** The Neural Geometric Fabrics [4], a structured policy class, that enables efficient skill learning for dexterous manipulation from demonstrations by leveraging structures induced by Geometric Fabrics [38]. Geometric Fabrics is a stable class of the Riemannian Motion Policy (RMP) [39]. It has been demonstrated that NGF outperforms RMP in policy learning for dexterous manipulation task in [4]. The NGF policy is defined in the configuration space of the robot, which is composed of a geometric policy, a potential policy and a damping term. More specifically, the NGF policy is constructed as follows: (1) define a geometric policy pair $[\mathbf{M}, \pi]$ and a potential policy pair $[\mathbf{M}_f, \pi_f]$ in the configuration space $\mathbf{q}$, (2) energize the geometric policy (project orthogonal to the direction of motion with $\mathbf{p}_e$) to create a collection of energy-preserving paths (the Geometric Fabric), and (3) force the Geometric Fabric with a potential defined by $[\mathbf{M}_f, \pi_f]$ and damp via $b$ applied along $\dot{\mathbf{q}}$, which ensures convergence to the potential's minima. The potential policy $\pi_f$ is the gradient of a function of position only. Note that we parameterize the geometric policy pair $[\mathbf{M}, \pi]$, the potential policy pair $[\mathbf{M}_f, \pi_f]$, and the damping scalar $b$ with MLP networks and learn them from demonstration data.

# F Optimizing baseline model size

As described in Appendices E, we sticked with the same set of lifting functions for KODex and report the task success rate when we trained KODex on training set and tested it on validation set in Table. 1. However, for baselines, the hyper-parameters were selected through a set of ablation experiments for each task using the training set over three choices of model size, including small size, median size and large size. We generated five random seeds for parameter initialization per model size, per baseline, and per task, as all learning based baseline models are sensitive to parameter initialization [23]. For each baseline policy, we report the mean and standard deviation of the task success rate on the validation set over five random seeds in Tables. 2-5.

Based on these results, we selectd the model size that offers the best performance in terms of task success rate. In addition, these results indicate that, unlike KODex, extensive hyper-parameter tuning and various trials on parameter initialization for baseline models are necessary. Note that we use $l$ to denote $\dim(\mathrm{x}(t))$.

Table 1: Task success rate on validation set (KODex)

| Tool | Door | Relocation | Reorientation |
|---|---|---|---|
| 100.0% | 96.0% | 88.0% | 62.0% |

Table 2: Hyper-parameters on NN Network Sizes

| Success Rate Task<br>(%)<br>Model Size | Tool | Door | Relocation | Reorientation |
|---|---|---|---|---|
| MLP: (32, 64, 32) | **0.4(±0.8)** | 0.0(±0.0) | 0.4(±0.8) | 6.8(±3.9) |
| MLP: (64, 128, 64) | 0.0(±0.0) | **0.4(±0.8)** | **1.2(±2.4)** | **10.4(±6.6)** |
| MLP: (128, 256, 128) | 0.0(±0.0) | 0.0(±0.0) | 0.8(±1.6) | 6.0(±1.5) |

Table 3: Hyper-parameters on LSTM Network Sizes

| Success Rate Task<br>(%)<br>Model Size | Tool | Door | Relocation | Reorientation |
|---|---|---|---|---|
| LSTM: 200<br>fc: ($l$, 100), (200, $l$) | 28.8(±25.0) | **87.6(±10.3)** | 7.6(±5.9) | **56.4(±7.4)** |
| LSTM: 250<br>fc: ($l$, 175), (250, $l$) | **60.8(±36.6)** | 80.8(±24.5) | 7.6(±7.5) | 48.0(±17.0) |
| LSTM: 300<br>fc: ($l$, 250), (300, $l$) | 44.8(±31.8) | 82.0(±13.9) | **16.4(±14.5)** | 54.0(±11.0) |

Table 4: Hyper-parameters on NDP Network Sizes

| Success Rate Task<br>(%)<br>Model Size | Tool | Door | Relocation | Reorientation |
|---|---|---|---|---|
| MLP: (32, 64, 32)<br>10 RBFs | 0.0(±0.0) | 8.0(±2.5) | 30.0(±9.3) | 57.2(±8.6) |
| MLP: (64, 128, 64)<br>20 RBFs | 16.8(±29.8) | 40.8(±8.1) | 74.0(±4.9) | 59.2(±6.5) |
| MLP: (128, 256, 128)<br>30 RBFs | **18.4(±31.9)** | **66.0(±5.2)** | **79.2(±7.7)** | **62.4(±7.8)** |

Table 5: Hyper-parameters on NGF Network Sizes

| Success Rate (%) Task Model Size | Tool | Door | Relocation | Reorientation |
|---|---|---|---|---|
| MLP: (64, 32) | 99.2(±1.6) | 87.2(±12.0) | 87.6(±8.5) | 77.6(±2.3) |
| MLP: (128, 64) | **100.0(±0.0)** | 90.0(±5.9) | 94.4(±3.2) | 72.4(±4.5) |
| MLP: (256, 128) | 83.6(±20.1) | **90.8(±4.3)** | **95.2(±1.6)** | **78.4(±3.4)** |

## G Hyper-parameters for controller learning

The hyper-parameters we used to learn the inverse dynamics controller $C$ for each task were the same as listed in Table. 6. Note that we use $l_r$ to denote $\dim(\mathrm{x}_r(t))$.

Table 6: Hyper-parameters on controller learning

| Hidden Layer | Activation | Learning Rate | Iteration |
|---|---|---|---|
| $(4l_r, 4l_r, 2l_r)$ | ReLU | 0.0001 | 300 |

## H Zero-Shot Out-of-Distribution Generalization

We generated a new set of 10,000 out-of-distribution samples to evaluate how the policies that were trained on 200 demonstrations generalize to unseen samples (see Appendices D for details on the sampling procedure). In Fig. 5, we report the task success rates of each method trained on the 200 demonstrations and tested on the 10,000 out-of-distribution samples. In addition, we also report the task success rate of the expert policy on the same 10,000 out-of-distribution samples to establish a baseline. Perhaps unsurprisingly, none of the methods are able to consistently outperform the expert policy in most tasks. We observe that KODex is able to outperform the four baselines in Tool Use task. In the other tasks, the highly-structured NGF performs the best, and KODex's performs comparably to NDP and LSTM.

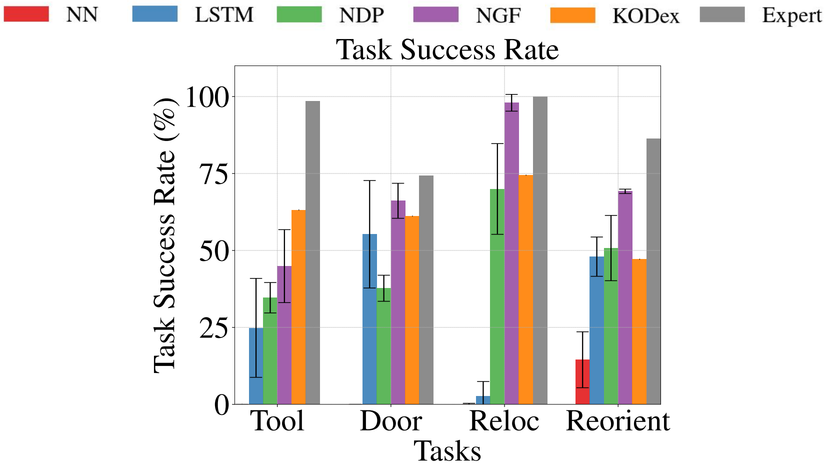


Figure 5: Zero-Shot Out-of-distribution task success rates

## I Robustness to changes in physical properties

We evaluate the robustness of the reference dynamics learned by each method to changes in hand mass or object mass for each task. This experiment is motivated by the fact that sim-to-real transfer often involves changes in physical properties. Further, consistent use of robotic hardware could result in changes to physical properties. Specifically, we consider four variations per task:

- Tool Use: i) *Heavy Object (Hammer)*: 0.25 (default) → 0.85 (new), ii) *Light Object (Hammer)*: 0.25 (default) → 0.10 (new), iii) *Light Hand (Palm)*: 4.0 (default) → 1.0 (new), and iv) *Heavy Hand (Palm)*: 4.0 (default) → 8.0 (new)
- Door: i) *Heavy Object (Latch)*: 3.54 (default) → 12.54 (new), ii) *Light Object (Latch)*: 3.54 (default) → 0.54 (new), iii) *Light Hand (Palm)*: 4.0 (default) → 1.5 (new), and iv) *Heavy Hand (Palm)*: 4.0 (default) → 7.0 (new)
- Relocation: i) *Heavy Object (Ball)*: 0.18 (default) → 1.88 (new), ii) *Light Object (Ball)*: 0.18 (default) → 0.05 (new), iii) *Light Hand (Palm)*: 4.0 (default) → 3.0 (new), and iv) *Heavy Hand (Palm)*: 4.0 (default) → 5.0 (new);
- Reorientation: i) *Heavy Object (Pen)*: 1.5 (default) → 9.5 (new), ii) *Light Object (Pen)*: 1.5 (default) → 0.2 (new), iii) *Light Hand (Finger Knuckles)*: 0.008 (default) → 0.0001 (new), and iv) *Heavy Hand (Finger Knuckles)*: 0.008 (default) → 0.20 (new)

It is important to note we held the reference dynamics learned by each method constant for this experiment, irrespective of the changes to the hand or the object. Instead, we relearned the tracking controller using 200 rollouts from the expert agent, following the procedure detailed in Section. 4.3.

In Tables. 7-10, we report the task success rate of KODex, and other baseline policies (all trained on 200 demonstrations) before and after relearning the controller. We also report the task success rates of the expert agents to establish baselines.

We find that the Light Hand variation results in the lowest drop in performance across all methods and all tasks, thus consequently relearning controllers does not offer any considerable improvements. In contrast, all methods benefit from relearning the controller in the Heavy Hand variations, as evidenced by the increased task success rates. Overall, we find that KODex outperforms all baselines, with the exception of NGF which performs better than KODex under a few variations and tasks. Surprisingly, KODex (and some baselines) when used with the original controller outperform the expert policy under a few variations (e.g., Heavy Object in Relocation task, and Heavy Object in Door task). We believe this is due to the fact that KODex and the baselines learn to generate and track desired trajectories separately, while the expert RL directly generates control inputs from state information. In particular, the learned desired trajectories for a given tasks are likely invariant to slight changes in physical properties. On rare occasions where this is not the case, we indeed find that fine-tuning the tracking controllers worsens the performance.

These results demonstrate that changes to the robot/system dynamics can be handled by fine tuning the tracking controller without the need for relearning the reference dynamics. Once again, KODex is able to perform comparably to or outperform SOTA approaches despite its simplicity.

Table 7: Robustness to variations in the physical properties (Tool Use)

| Success Rate Variation<br>(%)<br>Controller | Heavy Object | Light Object | Light Hand | Heavy Hand |
|---|---|---|---|---|
| Expert agent | 93.5 | 66.2 | 65.4 | 71.2 |
| KODex + Original controller | 46.0 | 64.0 | 99.5 | 46.5 |
| NN + Original controller | 0.0(±0.0) | 0.0(±0.0) | 0.0(±0.0) | 0.7(±1.4) |
| LSTM + Original controller | 32.7(±18.7) | 35.0(±22.1) | 44.3(±23.1) | 52.7(±27.5) |
| NDP + Original controller | 0.0(±0.0) | 68.0(±20.8) | 45.4(±37.4) | 0.0(±0.0) |
| NGF + Original controller | 33.4(±11.5) | 62.9(±27.5) | 83.2(±26.3) | 40.3(±20.2) |
| KODex + Expert-tuned controller | 53.5 | 44.0 | 89.0 | 92.5 |
| NN + Expert-tuned controller | 0.0(±0.0) | 0.0(±0.0) | 0.2(±0.4) | 0.0(±0.0) |
| LSTM + Expert-tuned controller | 42.4(±34.3) | 33.7(±14.9) | 52.2(±22.7) | 69.9(±19.4) |
| NDP + Expert-tuned controller | 33.3(±20.0) | 23.8(±24.4) | 29.4(±37.1) | 39.8(±24.5) |
| NGF + Expert-tuned controller | 48.2(±18.0) | 48.7(±12.2) | 94.6(±8.9) | 82.1(±7.5) |

Table 8: Robustness to variations in the physical properties (Door)

| Success Rate<br>Variation<br>(%)<br>Controller | Heavy Object | Light Object | Light Hand | Heavy Hand |
|---|---|---|---|---|
| Expert agent | 45.2 | 91.7 | 82.0 | 74.9 |
| KODex + Original controller | 57.0 | 97.0 | 56.5 | 33.5 |
| NN + Original controller | 0.0(±0.0) | 0.2(±0.4) | 1.3(±2.1) | 0.0(±0.0) |
| LSTM + Original controller | 34.4(±8.7) | 75.8(±19.5) | 38.1(±10.4) | 33.5(±11.4) |
| NDP + Original controller | 22.1(±1.9) | 62.8(±5.2) | 51.1(±4.9) | 3.1(±2.3) |
| NGF + Original controller | 48.7(±6.7) | 95.0(±2.1) | 42.1(±11.0) | 33.8(±10.0) |
| KODex + Expert-tuned controller | 39.0 | 94.0 | 54.0 | 81.5 |
| NN + Expert-tuned controller | 0.0(±0.0) | 0.0(±0.0) | 0.7(±0.9) | 0.0(±0.0) |
| LSTM + Expert-tuned controller | 21.2(±5.3) | 75.4(±18.0) | 49.2(±8.1) | 56.9(±18.7) |
| NDP + Expert-tuned controller | 15.5(±3.0) | 36.2(±10.6) | 25.5(±4.4) | 8.8(±3.0) |
| NGF + Expert-tuned controller | 36.6(±5.1) | 95.5(±1.8) | 57.7(±4.7) | 77.1(±6.7) |

Table 9: Robustness to variations in the physical properties (Relocation)

| Success Rate<br>Variation<br>(%)<br>Controller | Heavy Object | Light Object | Light Hand | Heavy Hand |
|---|---|---|---|---|
| Expert agent | 77.0 | 100.0 | 100.0 | 100.0 |
| KODex + Original controller | 19.5 | 89.5 | 82.5 | 21.5 |
| NN + Original controller | 0.1(±0.2) | 1.6(±2.5) | 1.5(±2.1) | 1.7(±2.2) |
| LSTM + Original controller | 0.4(±0.4) | 15.4(±10.7) | 9.5(±8.1) | 7.7(±9.4) |
| NDP + Original controller | 13.5(±5.0) | 85.6(±8.1) | 72.1(±9.6) | 31.6(±10.0) |
| NGF + Original controller | 25.8(±4.9) | 96.4(±1.4) | 96.6(±0.97) | 19.3(±3.8) |
| KODex + Expert-tuned controller | 34.0 | 93.0 | 85.0 | 89.0 |
| NN + Expert-tuned controller | 0.2(±0.4) | 0.6(±0.7) | 1.4(±1.8) | 1.5(±2.3) |
| LSTM + Expert-tuned controller | 5.8(±4.7) | 15.2(±12.5) | 15.5(±10.7) | 14.1(±9.3) |
| NDP + Expert-tuned controller | 19.9(±5.8) | 84.5(±8.9) | 63.2(±15.0) | 92.4(±1.2) |
| NGF + Expert-tuned controller | 52.6(±3.6) | 98.1(±1.2) | 95.6(±2.2) | 94.5(±0.9) |

Table 10: Robustness to variations in the physical properties (Reorientation)

| Success Rate<br>Variation<br>(%)<br>Controller | Heavy Object | Light Object | Light Hand | Heavy Hand |
|---|---|---|---|---|
| Expert agent | 46.8 | 69.0 | 95.2 | 89.7 |
| KODex + Original controller | 53.5 | 55.0 | 66.5 | 61.5 |
| NN + Original controller | 4.7(±2.6) | 9.6(±8.1) | 9.5(±6.4) | 7.9(±6.5) |
| LSTM + Original controller | 34.5(±7.8) | 52.3(±10.6) | 60.3(±6.0) | 55.6(±7.8) |
| NDP + Original controller | 49.4(±3.6) | 58.4(±6.4) | 59.8(±7.6) | 55.7(±9.7) |
| NGF + Original controller | 39.9(±1.9) | 57.1(±2.2) | 81.6(±1.8) | 73.4(±3.8) |
| KODex + Expert-tuned controller | 52.0 | 63.0 | 71.5 | 65.5 |
| NN + Expert-tuned controller | 1.5(±0.9) | 5.2(±4.2) | 3.8(±1.7) | 3.7(±2.6) |
| LSTM + Expert-tuned controller | 43.5(±7.9) | 47.7(±8.8) | 61.4(±4.2) | 54.4(±5.5) |
| NDP + Expert-tuned controller | 55.5(±5.9) | 59.0(±5.5) | 63.0(±6.5) | 57.0(±7.5) |
| NGF + Expert-tuned controller | 49.1(±2.6) | 59.7(±3.2) | 79.4(±1.9) | 72.6(±1.2) |

## J The impact of the choice of basis functions

We evaluate if KODex's performance is impacted by different sets of polynomial functions that are used as the lifting function. We trained all policies on 200 demos and tested them on 10,000 unseen initial conditions.

**Design**: Specifically, we define four sets of observables (one of which was used in the original submission). Let robot state: $\mathrm{x}_r = [x_r^1, x_r^2, \cdots, x_r^n]$ and $\mathrm{x}_o = [x_o^1, x_o^2, \cdots, x_o^m]$ denote the robot and the object state, respectively, with superscript indexing the states. We then define four vector-valued lifting functions $\psi_r$ and $\psi_o$ in (8) as follows

- Set 1

$$\begin{aligned} \psi_r &= \{(x_r^i)^2\} \text{ for } i = 1, \cdots, n \\ \psi_o &= \{(x_o^i)^2\} \text{ for } i = 1, \cdots, m \end{aligned}$$

- Set 2

$$\begin{aligned} \psi_r &= \{x_r^i x_r^j\} \text{ for } i, j = 1, \cdots, n \\ \psi_o &= \{x_o^i x_o^j\} \text{ for } i, j = 1, \cdots, m \end{aligned}$$

- **Set 3 (used in this work)**

$$\begin{aligned} \psi_r &= \{x_r^i x_r^j\} \cup \{(x_r^i)^3\} \text{ for } i, j = 1, \cdots, n \\ \psi_o &= \{x_o^i x_o^j\} \cup \{(x_o^i)^2 (x_o^j)\} \text{ for } i, j = 1, \cdots, m \end{aligned}$$

- Set 4

$$\begin{aligned} \psi_r &= \{x_r^i x_r^j\} \cup \{(x_r^i)^2 (x_r^j)\} \text{ for } i, j = 1, \cdots, n \\ \psi_o &= \{x_o^i x_o^j\} \cup \{(x_o^i)^2 (x_o^j)\} \text{ for } i, j = 1, \cdots, m \end{aligned}$$

We report the number of observables for each set and task combination in Table. 11.

Table 11: Number of observables

| Task<br>Set | Tool n=26,m=15 | Door n=28,m=7 | Relocation n=30,m=12 | Reorientation n=24,m=12 |
|---|---|---|---|---|
| Set 1 | 82 | 70 | 84 | 72 |
| Set 2 | 512 | 469 | 585 | 414 |
| **Set 3** (ours) | 763 | 546 | 759 | 582 |
| Set 4 | 1413 | 1302 | 1629 | 1134 |

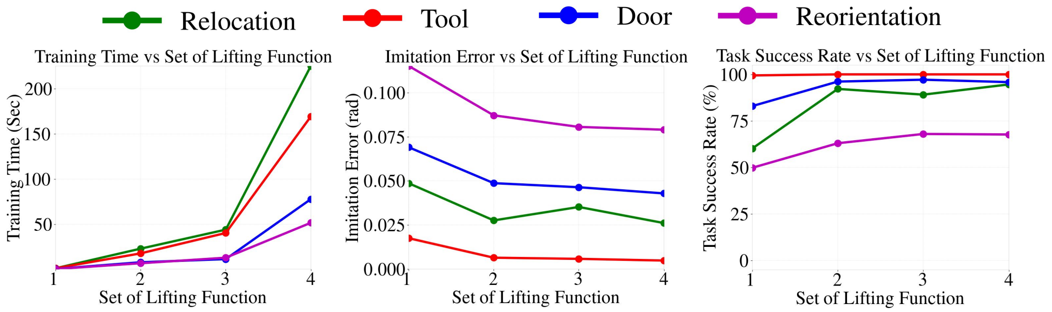


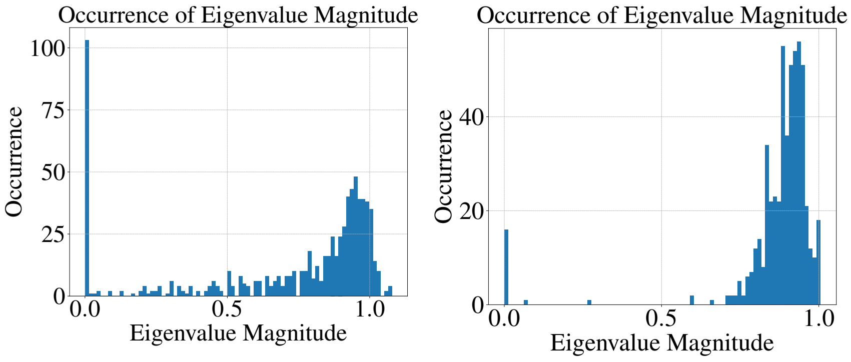
Figure 6: The effects of lifting function on training time (left), imitation error (center), and success rate (right).

**Discussion**: As shown in Fig. 6, it is clear that training time increases with the number of observables since the Moore–Penrose inverse requires more computation for higher-dimension matrices. Importantly, KODex's success rate across all tasks remained roughly the same for Sets 2, 3, and 4. In general, as one would expect, increasing the number of observables tends to decrease imitation error and increase task success rate. The only exception to this trend is observed for the Object Relocation task, in which KODex performs marginally better when trained on Set 2 (585 observables) compared with it trained on Set 3 (759 observables). Taken together, these results suggest that KODex's performance is not highly sensitive to the specific choice of lifting function, as long as sufficient expressivity is ensured.

# K Stability Analysis

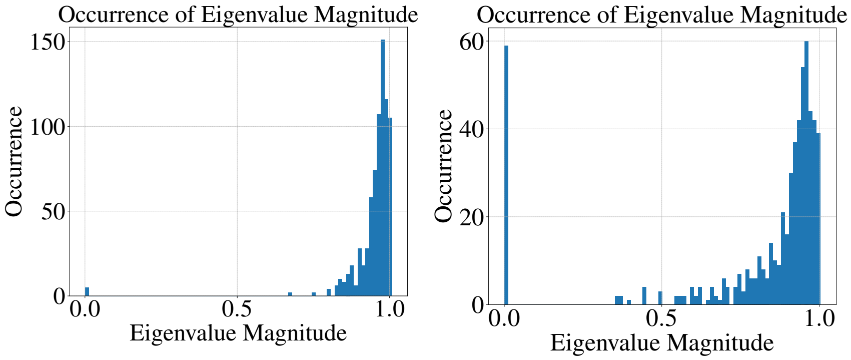
Another unique advantage of utilizing Koopman Operators to model the underlying dynamical system for dexterous manipulation tasks is that the learned policy is a linear dynamical system which can be readily inspected and analyzed, in stark contrast to SOTA methods built upon deep neural networks.

We analyzed the stability of the learned policy. For a linear dynamical system with complex conjugate eigenvalues $\lambda_i = \theta_i \pm j\omega_i$, i.e., KODex with Koopman matrix $\mathbf{K}$, the system is asymptotically stable if all of the eigenvalues have magnitude ($\rho_i = \sqrt{\theta_i^2 + \omega_i^2}$) less than one. From the standpoint of control theory, it is beneficial to have a asymptotically stable system because of the guarantee that all system states will converge. However, from the standpoint of dexterous manipulation tasks considered in this work, strict stability might not be preferable because the final desired hand poses and object poses are not identical for different initial conditions. This represents a natural trade-off between safety and expressivity. As such, understanding how KODex addresses this trade-off can be illuminating.



(a) Tool Use

(b) Door Opening



(c) Object Relocation

(d) In-hand Reorientation

Figure 7: Occurrence of Eigenvalue Magnitude

Table 12: Maximun Eigenvalue Magnitude

| Tool Use | Door Opening | Object Relocation | In-hand Reorientation |
|---|---|---|---|
| 1.07888 | 1.00553 | 1.00859 | 1.00413 |

In Fig. 7, we report a histogram of the Koopman matrix's eigenvalue magnitudes in each task. In addition, we report the maximum eigenvalue magnitude in Table. 12. Based on these results, we can see that i) most eigenvalues' magnitudes are less than one, suggesting that KODex tends to learn nearly-stable policies that generate *safe trajectories* during execution, and ii) a few eigenvalues have magnitude larger than one, suggesting KODex does not prioritize stability, at the expense of expressivity required to achieve the reported performance.