Extended Abstract

Motivation Wire sorting is a useful but difficult industrial robotics task: a robot must separate braided wire ends and align each wire to a target location and orientation. Directly simulating and manipulating deformable wires is beyond the scope of the class project, so we study a rigid proxy problem in Robosuite: a Franka Panda must rearrange one or two thin rod-like objects on a tabletop. The proxy preserves several important structural problems from wire sorting, including long horizons, repeated pick-and-place subskills, object ordering, and contact-sensitive grasp and placement errors.

Pipeline We built a full learning pipeline around this proxy task. A scripted pick-and-place oracle generates successful demonstrations using privileged simulator state. The oracle follows an eight-phase finite-state controller: approach, descend, grasp, lift, transport, place, release, and retreat. From these demonstrations, we train behavior-cloning policies: a single-step MLP-BC policy and a diffusion/flow-matching behavior-cloning (DPFM-BC) policy that predicts short action chunks. We then train residual TD3 on top of a frozen DPFM policy. At each timestep, the base policy proposes a normalized action, the residual actor predicts a bounded correction, and the environment executes the sum after denormalization.

Implementation The environment state is a flat state vector containing robot proprioception, end-effector pose, stick poses, and goal positions. The action is a 7-D OSC_POSE command: end-effector translation, rotation, yaw, and gripper command. The one-stick task uses a 60-D state vector s and the two-stick task uses a 70-D state. Phase-conditioned policies append a one-hot phase label and a one-hot active-stick label to the environment state s. Demonstrations are stored in reproducible HDF5 files with observations, actions, rewards, next observations, done flags, success flags, phase labels, active-stick labels, random seed, environment hash, Robosuite version, and oracle version.

Results We collected 200 successful N=1 demonstrations, 200 successful N=2 fixed-order demonstrations, 200 successful N=2 balanced random-order demonstrations, and an N=2 paired-order dataset for multimodality analysis. The N=2 fixed-order MLP baseline exposed a context problem: observation-only MLP-BC achieved near-zero success, while adding oracle phase and active-stick labels reached about 94%. Balanced random-order data reduced this shortcut and raised observation-only MLP-BC to the 60–70% range. The paired-order dataset then created true same-state multimodality, where DPFM-BC reached 66% success while MLP-BC fell to 34%, motivating DPFM as the base for online residual learning.

Residual TD3 policy improved over the DPFM baseline on both tasks. On N=1, the residual policy improved success rate from 80–87% DPFM baseline to 99%. The same RRL approach was then applied to N=2: a critic ensemble consisting of 4 networks, 3-step returns, mixed offline/online replay buffers, and bounded residuals in normalized action space. The final N=2 residual run used `action_scale`=0.1 and 50% offline replay, improving the success rate of the observation-only DPFM baseline from 60% to 75% over 100 evaluation episodes. A demo-seeded SAC baseline reached only 33% success on N=1, supporting the choice to learn corrections on top of a strong cloned base rather than learning the full policy from scratch.

Discussion The results support three main conclusions. First, high-level task context matters: the policy must know not only how to grasp and place a stick, but which phase and which object are currently active. Second, context must be matched to the policy architecture. Phase labels are very effective for single-step MLP-BC, but they create a train-time mismatch for DPFM when target chunks cross phase boundaries. Third, residual RL was able to further improve DPFM on N=1 and N=2, showing that online interaction can correct systematic errors left by imitation learning.

Conclusion This project demonstrates a practical BC-to-residual-RL pipeline for a long-horizon robotic manipulation proxy task. We did not solve deformable wire manipulation directly, but we developed the environment, demonstration collection procedure, behavior-cloning baselines, phase/context diagnostics, and residual TD3 training needed for future work on more realistic wire-like objects and larger object counts.

Residual Reinforcement Learning for Robotic Manipulation of Wire-like Objects

Bautista Guerra
Stanford University
baguerra@stanford.edu

Alexander Tarvo
Stanford University
alexta@stanford.edu

Andrew Yuxuan Liang
Stanford University
aliang29@stanford.edu

Abstract

We study robotic manipulation of wire-like objects through a rigid-stick proxy task in Robosuite. A Franka Panda robot must move one (N=1 task) or two (N=2) thin rods from randomized initial poses to goal positions and orientations. We implement a scripted demonstration oracle, collect reproducible HDF5 datasets, train behavior-cloning policies, and train residual TD3 on top of a frozen diffusion/flow-matching behavior-cloning policy. The demonstration oracle uses an eight-phase finite-state controller and logs phase and active-stick labels, enabling observation-only and oracle-context policy variants. Residual TD3 improves the selected DPFM-BC baseline from roughly 80–87% to 99% success on N=1 and from 60% to 75% success on N=2. We also find that balanced demonstration ordering substantially improves observation-only N=2 behavior cloning, that same-state multimodal demonstrations favor DPFM over deterministic MLP-BC, and that phase-active context helps single-step MLP policies but is harder to use with chunked DPFM policies.

1 Introduction

Robotic manipulation of wire-like objects is an important capability for robotic assembly across a range of industries, including aerospace and shipbuilding. In particular, a robot must separate wire ends from a bundle and align them to target locations. The problem requires multi-stage ordering, dexterous manipulation of thin contact-sensitive objects, and simulation of deformable wires, which is difficult in standard rigid-body engines.

Therefore, we study a simplified but structured proxy task. Instead of deformable wires, the environment contains one (N=1) or two (N=2) rigid rods on a tabletop. Each rod has a randomized initial pose and a target position with desired yaw alignment. The robot must pick up each rod, transport it, place it, and release it. This proxy removes deformable-body physics while preserving the long-horizon structure of wire sorting: repeated phases, object identity, ordering, grasp timing, and precise placement.

Manipulation of multiple objects is a complex robotics problem that unifies long-term planning, elaborate sensing, and precise object manipulation. Implementing a traditional control-based robotics algorithm for wire manipulation would require significant effort, lack flexibility and adaptability to new task configurations, and thus is unlikely to succeed. Instead, we explore modern methods for robotic control, such as behavior cloning (a kind of imitation learning) and reinforcement learning.

Behavior cloning algorithms alone are sensitive to distribution shifts between training and deployment phases; they are vulnerable to compounding errors, which limits their task success rate in practical deployment. Pure online RL is inefficient for long-horizon, contact-rich manipulation. Therefore, we rely on residual reinforcement learning, which bootstraps reinforcement learning from demonstrations.

We perform our experiments in a simulated environment that mimics the manipulation of stick-like objects with the Franka Panda robotic arm. The environment allows configuring the number of sticks placed at random locations on a table; it collects the robot's proprioceptive state and controls the robotic arm using an OSC_POSE-integrated controller. The environment collects the precise positions of a robotic arm and objects, enabling the collection of expert demonstrations using the Inverse Kinematics (IK) policy.

We use the collected demonstrations to train two versions of the behavior-cloning (BC) policies. A simple MLP-BC policy is based on the multilayer perceptron and uses an L2 loss to learn the action a^b based on the vector of current observations o as $a^b = \pi_{bc}(o)$. A more elaborate DPFM policy uses the diffusion flow matching policy based on 1D convolution network, which effectively handles multi-modal distributions of the action a^b for a given observation o.

Finally, we train a residual reinforcement learning (RRL) control policy π_r based on the ResFiT algorithm. ResFit treats the underlying BC policy π_{bc} as a "black box". It learns a small residual action correction vector $a^r = \pi_r(o, a^b)$. Then the RRL algorithm computes the final action as $a = a^b + \alpha a_r$, where α is a hyperparameter that controls the weight (or scale) of the residual action.

We evaluate our approach using two main evaluation metrics: the task completion rate and the average trajectory reward. We demonstrate that the integration of the high-level context information, such as an active task execution phase, significantly improves the performance of simpler behavior cloning policies. At the same time, the residual RL achieves the biggest improvement for the object manipulation tasks.

Our main contributions are:

- A comparison of MLP-BC and DPFM-BC behavior-cloning baselines under observation-only, phase-active, random-order, and paired-order data settings.
- A residual TD3 implementation that improves over a frozen DPFM policy by adding bounded residual actions in the normalized action space.
- A SAC, Haarnoja et al. (2018), baseline showing that direct full-action RL remains difficult even when seeded with the same expert demonstrations.
- An analysis of why phase labels help single-step policies but can hurt chunked diffusion/flow-matching policies near phase boundaries.

2 Related Work

Behavior cloning is a common first stage for robot learning because demonstrations can provide a strong initialization for long-horizon tasks. Modern visuomotor imitation methods, such as the Diffusion Policy model multimodal action distributions through denoising or flow-matching, Chi et al. (2024). In our project, we adapt this idea to a state-based setting by training a temporal diffusion/flow-matching policy over short action chunks.

Behavior cloning alone can fail when the learner visits states outside the demonstration distribution. Residual reinforcement learning addresses this by learning corrective actions on top of a base controller or behavior-cloned policy. Johannink et al. (2019) introduced residual RL for robot control, where a learned policy augments an existing controller. Recent residual fine-tuning work shows that keeping the base policy fixed can preserve useful prior behavior while allowing online improvement, Ankile et al. (2024, 2025). Our method follows the same principle: DPFM remains frozen and TD3 learns only the corrective residual.

We use TD3 because deterministic actor-critic methodswell-suiteduited to continuous control and include stabilizers such as clipped double-Q targets, delayed actor updates, target networks, and target-policy smoothing, Fujimoto et al. (2018). Our implementation extends the clipped double-Q idea to a critic ensemble: Bellman targets use the minimum over a sampled subset of critics, while actor updates use the mean over the current heads.

3 Task and Environment

We solve the task of manipulating small, elongated objects – sticks -— that lie on a table. Each stick has a length of about 0.20 m and a radius of 0.0075 m. They must be re-positioned at the center of the table as demonstrated in Figure 1. The environment supports multiple stick positions N; currently, we experiment with N=1 and N=2. In the two-stick task, the blue stick starts on the negative-Y side of the table, and the red stick starts on the positive-Y side.

In our simulation, we employ the Franka Panda robotic arm to place the sticks at the target locations. The task is considered successfully completed once all the sticks are placed within their target limits within a time horizon of 500 environment steps; otherwise, we consider the robot to have failed to complete the task.

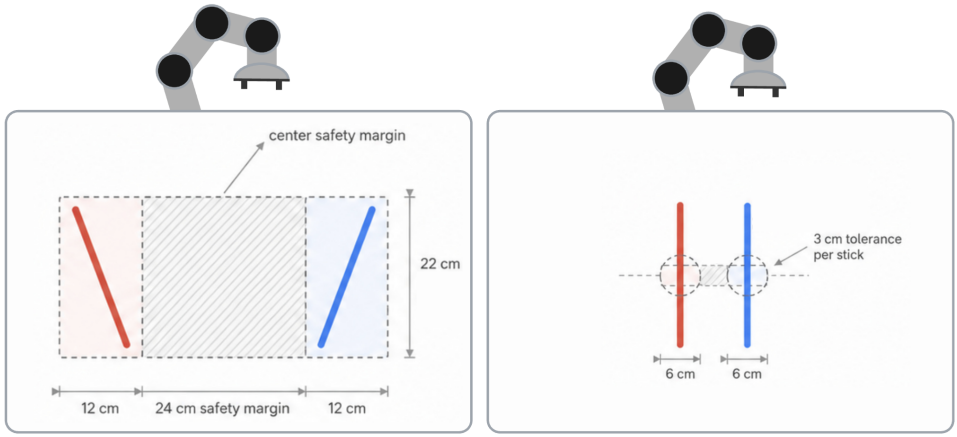

Figure 1: Two-stick task geometry. Initial stick positions are sampled from side-specific regions, while target positions lie in narrower side-specific goal bands. The balanced random-order dataset keeps this geometry fixed but alternates which stick the expert solves first.

We traditionally formulate the problem as a Markov decision process with state s_t, action a_t, and reward r_t.

The state is a variable-length vector that concatenates the states of the robot and the objects (sticks). The robotic state is a vector of robot proprioception $\in \mathcal{R}^{50}$. The stick state includes the initial location and orientation, and its goal position. Locations are represented by Cartesian coordinates and orientation by a quaternion $q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$, resulting in the final state vector for a stick $\mathbf{p} = (x, y, z, q_w, q_x, q_y, q_z) \in \mathbb{R}^3 \times S^3$. Thus, the one-stick observation vector has 60 dimensions, and the two-stick observation vector has 70 dimensions. In a subset of our experiments, we employed phase conditioning for the task. Phase-active conditioning appends an 8-D one-hot vector that defines the current phase (see below for the description of the phases) and an N-D one-hot vector that denotes the currently handed stick.

The action is a 7-D OSC_POSE command:

$$a_t = (\Delta x, \Delta y, \Delta z, \Delta roll, \Delta pitch, \Delta yaw, g),$$

where g is the gripper command. The controller maps translational commands to approximately ± 0.05 m per step and yaw commands to approximately ± 0.5 rad per step.

The shaped reward is the negative sum of stick position error plus a yaw-error penalty, with a large $+50$ success bonus:

$$r_t = -\sum_{i=1}^{N} (\|p_i - p_i^\star\|_2 + \lambda_{\text{rot}} e_i) + \mathbf{1}[\text{success}] \cdot R \tag{1}$$

where $\lambda_{\text{rot}} = 0.1$ and e_i is yaw error modulo the stick's 180-degree symmetry, and $R \in \mathcal{R}$ is the final reward upon a successful rollout. A rollout is successful when every stick is within 0.03 m of its goal and within 0.1745 rad of the desired yaw.

4 Demonstration Collection

Instead of using human teleoperation, we built a scripted pick-and-place oracle that uses privileged simulator state: end-effector pose, stick pose, and goal pose.

We define our "expert" IK-based policy as a state machine with eight phases $\mathcal{P} = \{\text{Approach}, \text{Descend}, \text{Grasp}, \text{LiftTransport}, \text{PlaceRelease}, \text{Retreat}\}$ that executes a pick-and-place trajectory using proportional control over OSC_POSE deltas.

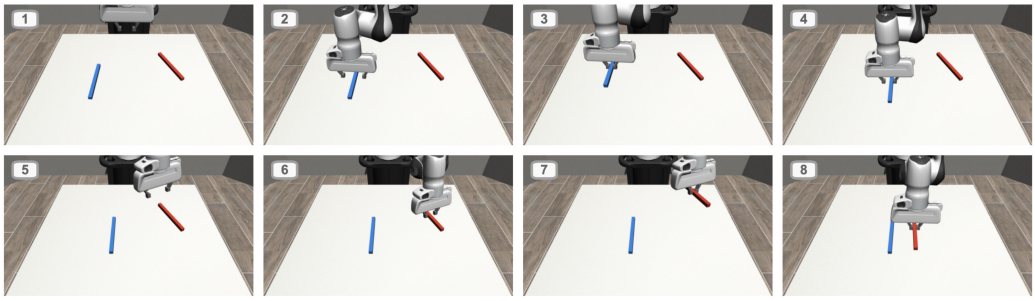

Figure 2: Example N=2 scripted rollout. The oracle grasps and places one stick, retreats, then repeats the same finite-state sequence for the second stick.

We keep only successful episodes for training. Failed attempts are discarded by default or can be saved separately for diagnostics. Each saved transition contains $(s_t, a_t, r_t, s_{t+1}, done)$ plus a success flag, phase label, and active-stick label. The root HDF5 metadata records the top-level seed, environment configuration hash, Robosuite version, oracle version, and observation index map, enabling observation-only and phase-active variants from the same trajectories.

Table 1: Demonstration datasets used in the project. All rows contain successful scripted rollouts.

Dataset	Demos	Transitions	Obs. dim	Notes
N=1 single stick	200	∼22k	60	one object
N=2 fixed order	200	∼49k	70	blue first, red second
N=2 random order	200	∼49k	70	balanced 50/50 order
N=2 paired order	200	∼49k	70	same reset, both orders

5 Method

We train two behavior-cloning (BC) policies on collected demonstrations: MLP-BC and DPFM-BC.

MLP-BC is a multi-layer perceptron (MLP) that, for the time step t predicts a single action a_t. MLP-BC consists of three hidden layers of 256 units each with ReLU activations, followed by a $\tanh$ output layer that constrains predictions to the $[-1, 1]$ range, as expected by the OSC_POSE controller.

The network is trained to minimize the mean-squared error between predicted action and the expert action a_i^*:

$$\mathcal{L}_{\text{MLP-BC}} = \frac{1}{B} \sum_{i=1}^{B} \|\pi_\theta(s_i) - a_i^*\|^2 ,$$

DPFM-BC is a diffusion policy based on conditional flow matching Lipman et al. (2023); Chi et al. (2024). DPFM-BC learns a velocity field $v(a_t^\tau; s_t, \tau)$ that generates an action a_t by iterative Euler integration with the step τ, advancing from a zero-mean unit-deviation Gaussian $a_t^0 = \mathcal{N}$ to the conditional action distribution $a_t(s_t)$.

The velocity network v_θ is a 1-D temporal U-Net adapted from Chi et al. (2024). The input noisy action chunk $\mathbf{a}_\tau \in \mathbb{R}^7$ and a flow timestep $\tau \in [0, 1]$ are encoded via sinusoidal positional embeddings.

The training loss is defined as:

$$\mathcal{L}_{\text{DPFM}} = \mathbb{E}_{\tau\sim\mathcal{U}(0,1),\ \mathbf{a}_0\sim\mathcal{N}(0,I)}\left[\left\|v_\theta(\mathbf{a}_\tau, s_t, \tau) - \mathbf{u}\right\|^2\right].$$

DPFM-BC policy predicts a horizon of H future actions $(a_t, \ldots, a_{t+H-1})$ – an *action chunk* – based on the current state s_t. During execution, the first E actions of the chunk are executed before the policy is called to produce the next chunk. We denote configurations as `hH_eE_iI`, where H is the prediction horizon, E is the chunk length, and I is the number of Euler integration steps used at inference time. DPFM policy represents multimodal action distributions better than a MLP-based regression.

During the training and inference, both actions and input observations are normalized per dimension using statistics computed over the training corpus. These statistics are saved with the checkpoint so that the reverse de-normalization is applied at inference time. We experimented with z-normalization, where datapoints have a zero mean and a unit standard deviation, as well as min/max normalization.

5.1 Residual TD3

Our implementation of the RRL is based on a ReSiFit work by Ankile et al. (2024). Residual RL (RRL) is trained on top of a frozen DPFM-BC checkpoint. Let a_t^b be the base policy action in normalized action space. The residual actor receives both the state and base action and outputs a bounded correction:

$$a_t^r = \pi_\theta([s_t, a_t^b]), \qquad a_t = a_t^b + \alpha a_t^r,$$

where α is the residual action scale. The combined normalized action a_t is then de-normalized by the base policy's action normalizer and executed in the environment.

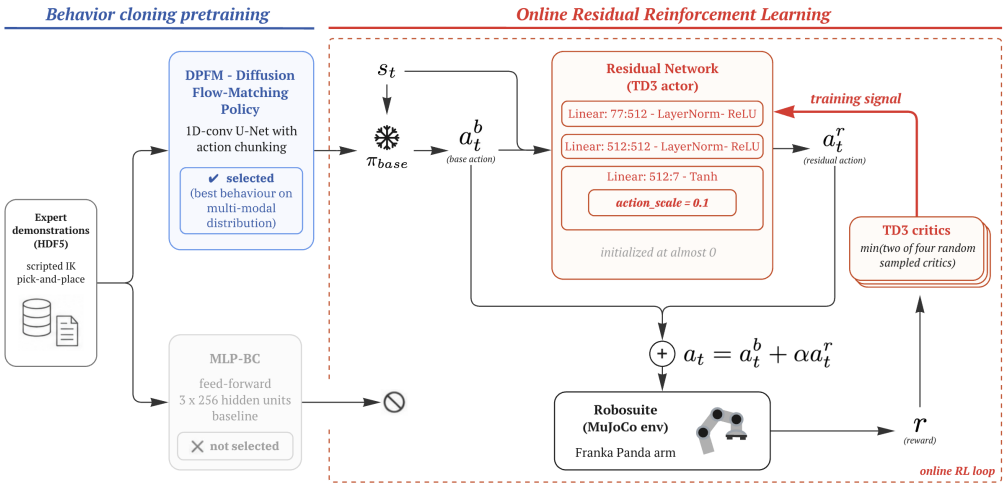

Figure 3: Training pipeline.

We use the TD3 RL algorithm as a base reinforcement learning algorithm. TD3 is an off-policy RL algorithm that maintains the actor π_θ and critic π_ϕ networks, as well as their EMA targets π'_θ and π'_ϕ. TD3 takes a single environment step during each iteration and stores the resulting transition into the $\mathcal{D}_{online}$ buffer.

We maintain a pair of online $\mathcal{D}_{online}$ and offline $\mathcal{D}_{offline}$ buffers, where $\mathcal{D}_{offline}$ contains expert demonstrations. Each training batch B contains 512 transitions; following RLPD Ball et al. (2023), it mixes 50% offline demonstration transitions and 50% online rollout transitions from a replay buffer of capacity 2×10^5.

The residual actor $\pi_\theta(s_t, a_t^b)$ is the MLP network with two 512-unit hidden layers, each followed by LayerNorm and ReLU. The top layer of π_θ has a tangential activation, s.t. the action dimensions remain within the [-1; +1] bounds. The critic is an ensemble of four Q-networks, each with the same two-layer architecture (512 units, LayerNorm, ReLU); mapping $(s_t, a_t) \mapsto Q(s_t, a_t) \in \mathbb{R}$. For

computing the Bellman target, we randomly sample two of the four critics and take the minimum predicted value to ensure training stability.

We used a 3-step returns to compute the critic target

$$y_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \min_{j \in \mathcal{J}} Q_{\phi'}(s_{t+n}, a_{t+n}),$$

$\mathcal{J}$ is a random two-head subset of the four target critics. We define the discount $\gamma = 0.97$ and EMA constant $\tau = 0.005$. The critic is updated 4 times per environment step at a learning rate of 10^{-4}; the actor is updated once per step at a learning rate of 5×10^{-6}, with the first 15,000 steps reserved for a critic-only warm-up.

Phase-active conditioning. When manipulating multiple sticks, the robot manipulates one stick at a time, cycling through eight pick-and-place phases $\mathcal{P}$ for each stick. We hypothesize that conditioning the policy on the index of the stick being currently manipulated $\mathbf{e}_k$, and the phase $\mathbf{e}_p$ will facilitate training. Thus we augment the current state s_t with $\mathbf{c}_t = [\mathbf{e}_{p_t}; \mathbf{e}_{k_t}]$, where $\mathbf{e}_{p_t} \in \{0,1\}^8$ is a one-hot phase indicator and $\mathbf{e}_{k_t} \in \{0,1\}^N$ a active-stick indicator.

During the data collection, $[\mathbf{e}_{p_t}; \mathbf{e}_{k_t}]$ is provided by the expert policy. At inference time, a copy of the scripted expert runs alongside the learned policy as a *phase tracker*. Its eight-phase state machine transitions are driven entirely by geometry heuristics based on the environment state —for example, Approach advances to Descend when the end-effector is within 2 cm of the target stick and the gripper yaw is aligned. Because these predicates depend only on positions and orientations read from s_t, the tracker acts as a primitive state-phase classifier, independent of which policy produces the actions. Computed p_t and k_t are concatenated with the state vector $\tilde{s}_t = [s_t; \mathbf{c}_t]$ and passed to the MLP-BC or DPFM-BC policy.

Although such a phase tracker produces approximate phase information, its heuristic-based design may produce phases that deviate substantially from expert-like behavior. As a result, the tracker's geometric predicates may trigger phase transitions that do not reflect the true task state, producing incorrect conditioning labels. A more robust learning-based phase detector, i.e., one based on an LSTM network, remains a subject of future work.

6 Results

We evaluate our models in the following configurations:

- **N=1** one randomized stick must be placed at a goal position and orientation.
- **N=2** two sticks must be placed at corresponding goal positions on the table. This is a more challenging configuration that involves the more diverse movement patterns (i.e. withdrawal of an arm) as well as longer task horizon

Furthermore, we varied the expert behavior for N=2 case:

- **fixed order:** the expert and phase tracker always handle the blue stick first and the red stick second.
- **random order:** the expert randomly selects blue or red as the first stick to manipulate;
- **paired multimodal order:** For each random placement of sticks, the expert performs two demonstrations, selecting the red stick first and then the blue stick first. This behavior emphasizes the bi-modal nature of the task;

All evaluations report closed-loop success over 100 episodes unless otherwise stated.

6.1 Behavior Cloning, Context, and Multimodality

To study the effect of the expert demonstrations and phase conditioning, we trained MLP-BC on 200 expert trajectories in N=1 and N=2 configurations. Table 2 summarizes the results for behavior-cloning policies. Here, "obs" denotes that the policy was conditioned only on the state vector s; "obs, phase" denotes conditioning the policy on the vector $[s_t; \mathbf{c}_t]$ that contains both the state as well as the inferred phase and the active stick ID.

A single-stick task, N=1, is largely solved by both behavior-cloning families. However, N=2 instead exposes subtask interference and dataset-structure issues.

Table 2: Closed-loop behavior-cloning results from 100-episode evaluations.

			$N = 1$		$N = 2$	
Method	Expert	Conditioning	Success	Return	Success	Return
DPFM-BC	random	obs	82%	-37.2 ± 34.8	60%	-124.3 ± 87.1
MLP-BC	random	obs	94%	-27.9 ± 31.7	63%	-117.7 ± 95.6
DPFM-BC	random	obs, phase	–	–	61%	-123.8 ± 85.1
MLP-BC	random	obs, phase	–	–	90%	-68.5 ± 55.4
DPFM-BC	paired multimodal	obs, phase	–	–	66%	-126.2 ± 95.8
MLP-BC	paired multimodal	obs, phase	–	–	34%	-253.3 ± 126.3

We observe that phase conditioning, although primitive, improves the success rate of the MLP-BC policy. This is especially evident for the fixed-order N=2 MLP ablation, not included in the table: observation-only MLP-BC achieved only 0–1% success, while phase-active MLP-BC reached about 94%. In that dataset, stick identity, table side, and temporal role were perfectly correlated. Balanced random-order demonstrations broke this shortcut and raised observation-only MLP-BC into the 60–70% range.

The paired-order dataset tested a different failure mode: the same initial observation has two valid expert trajectories, one solving blue first and the other red first. Deterministic MLP-BC tends to average incompatible actions, resulting in a low success rate of 34%. DPFM-BC represents a bi-modal distribution over action chunks, achieving a 66% success rate on paired data. This motivates the choice of DPFM as the base policy for residual RL.

Throughout these experiments, we demonstrated that adding phase conditioning significantly improves the performance of the MLP-BC policy. We hypothesize two reasons for that: first, the behavior of the MLP-BC is the closest to the demonstrations, so the phase tracker provides the most accurate predictions; second, phase tracking helps resolve possible multi-modalities in the state-action distributions to which the MLP-BC is naturally susceptible. At the same time, we observed only modest improvements in the DPFM performance from the phase tracker.

6.1.1 DPFM Phase-Context Diagnostics

Phase-active DPFM exposed a policy-architecture mismatch. Phase-active conditioning is straightforward for one-step MLP-BC because each input label supervises one action:

$$(s_t, \mathbf{e}_{p_t}, \mathbf{e}_{k_t}) \mapsto a_t,$$

For DPFM, the same label conditions an entire action chunk:

$$(s_t, \mathbf{e}_{p_t}, \mathbf{e}_{k_t}) \mapsto [a_t, a_{t+1}, \ldots, a_{t+H-1}].$$

Near phase boundaries, the target chunk can contain actions generated by later phases even though the input context is still the earlier phase. For `h8_e4`, 15.8% of executed prefixes and 36.9% of full training horizons crossed a phase boundary in the two-stick datasets. Reducing cached execution and shortening the horizon improved phase-active DPFM, but did not close the gap to phase-active MLP-BC. Boundary-aware chunking and padded-tail masking improved some stale-context cases, but the best random-order phase-active DPFM result remained below the simpler observation-only DPFM baseline.

6.2 Residual RL

Residual training was first implemented on N=1 (see Table 3). The base policy was DPFM in `h8_e4_i20` configuration, i.e. planning horizon of 8 steps, 4 steps executed before re-planning, 20 integration steps. For N=1 runs, we maintained two types of rewards: a single sparse reward $R = 50$ assigned upon successful task completion; and a dense reward computed according to the formula 1.

RRL has significantly improved performance on N=1, achieving 97-99% success rate. Interestingly, adjusting hyperparameters such as action scale α and update-to-data (UTD) ratio did not significantly

Table 3: Residual RL results for N=1.

Reward Type	Action Scale α	UTD ratio	Trained Steps	Episodes	Success Rate	Mean Reward	Residual L1 norm
dense	0.3	4	466000	4940	97.0%	38.23	0.259
dense	0.2	4	467000	4686	99.0%	40.28	0.176
dense	0.2	1	921000	8602	97.3%	38.44	0.173
sparse	0.2	4	436000	3367	95.0%	47.50	0.180

Figure 4: Enter Caption

affect policy performance; only the sparse reward decreased the success rate to 95%. In comparison, a pure Soft-Actor-Critic algorithm from the Sparse-Baselines3 library Raffin et al. (2021) reached the peak success rate of 33% (see Table 4).

Figure 5 shows the key statistics of the best N=1 residual run trained with the final reward $R = 50$. The mean episode reward steadily increases throughout training; actor loss begins to decrease after the initial exploration phase, while Q-values simultaneously increase – indicating that the policy is learning the correct residual actions.

However, the residual actions histogram demonstrates a worrisome trend. They quickly converge to a strongly bimodal distribution, whose modes are concentrated at the boundaries of the acceptable range $[-\alpha, \alpha]$. Despite a strong performance of a residual RL policy, this behavior warrants further investigation.

Figure 5: Key statistics for the N=1 residual run. Top row: mean episode reward, residual action L1 norm. Middle row: critic loss, actor loss. Bottom row: mean target Q-value, residual action histogram.

N=2 proved to be a more challenging configuration for both the base policy and RRL (see Table 4). We used the observation-only DPFM `h4_e1_i10` for the frozen base, as it demonstrated superior performance compared to `h8_e4_i20` for N=2. Still, the completion rate for the base policy was only 60%, and RRL further improved it to 75%. Figure 6 shows that `action_scale`=0.1 with 50% offline

Table 4: Aggregated results for Residual RL

Policy	Task	Success	Special notes and parameters
SAC	N=1	33%	Pure RL method; no base policy
DPFM-BC	N=1	85%	Behavior clone base policy; obs-only, H=8, E=4, I=20
DPFM + residual TD3	N=1	99%	$\alpha = 0.2$, $\lvert\mathcal{D}_{offline}\rvert = 0.5\lvert B\rvert$
DPFM-BC baseline	N=2	60%	Behavior clone base policy; obs-only, H=4, E=1, I=10
DPFM + residual TD3	N=2	75%	$\alpha = 0.1$, $\lvert\mathcal{D}_{offline}\rvert = 0.5\lvert B\rvert$

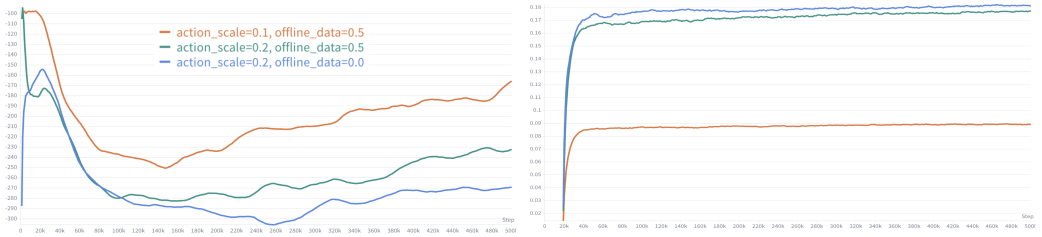

Figure 6: N=2 residual sweep over residual scale and offline replay. The selected run, `action_scale`=0.1 with 50% offline replay, recovers higher reward while using a smaller residual range than the larger-scale alternatives.

replay recovered higher reward than larger-scale or online-only alternatives while using a smaller residual range.

Figure 7 visualizes a successful rollout of the RRL policy for N=2. The critic value rises as the rollout progresses and the two sticks are placed. The critic advantage, $Q(a_{\text{final}}) - Q(a_{\text{base}})$, is usually positive, meaning the learned critic generally prefers the residual-corrected action over the frozen DPFM action. Together with the residual magnitude trace, this supports the interpretation that residual TD3 is compensating for systematic DPFM errors throughout the trajectory.

7 Discussion

In this work, we explored two main approaches to robotic manipulation: phase-conditioned behavior cloning and residual reinforcement learning.

For behavior cloning, our main empirical finding is the *importance of a high-level task structure*, especially with relatively simpler policies such as MLP-BC. MLP-BC proved to be surprisingly accurate for manipulating a single object (i.e. N=1 task), where the action distribution is unimodal. However, the N=2 task yields a multimodal action space that implicitly requires the policy to infer which object is active and which subskill to execute. Phase-active labels provided by the oracle explicitly establish a high-level task structure and simplify MLP-BC training, while balanced random-order demonstrations reduce shortcut learning.

Although a single current-phase label is appropriate for a single-step policy, it can be mismatched with a chunked policy when the chunk spans a phase transition. This could explain why the phase-active DPFM underperformed, despite the phase-active MLP-BC performing well. At the same time, DPFM proved robust in the paired-order multimodal task, where the deterministic MLP-BC averaged incompatible first actions. These two observations point to the same principle: the policy architecture and context interface must match the structure of the demonstrations.

Residual RL is a relatively novel approach to robot learning that is currently underutilized in industrial robotics. In fact, the robotics community remained divided at ICRA'2026: although some experts acknowledge the value of residual RL for object manipulation in real-world settings, others consider it overkill and propose focusing on learning-from-demonstration techniques, such as behavior cloning.

In our setting, RRL significantly improved the final DPFM baseline for both N=1 and N=2 objects. Compared to the pure-RL SAC baseline, RRL yielded both a higher success rate and shorter training times. At the same time, our experiments highlighted the problems of RRL. First, RRL proved to be a much less stable method than DBFP, sensitive to hyperparameters such as the actor/critic learning

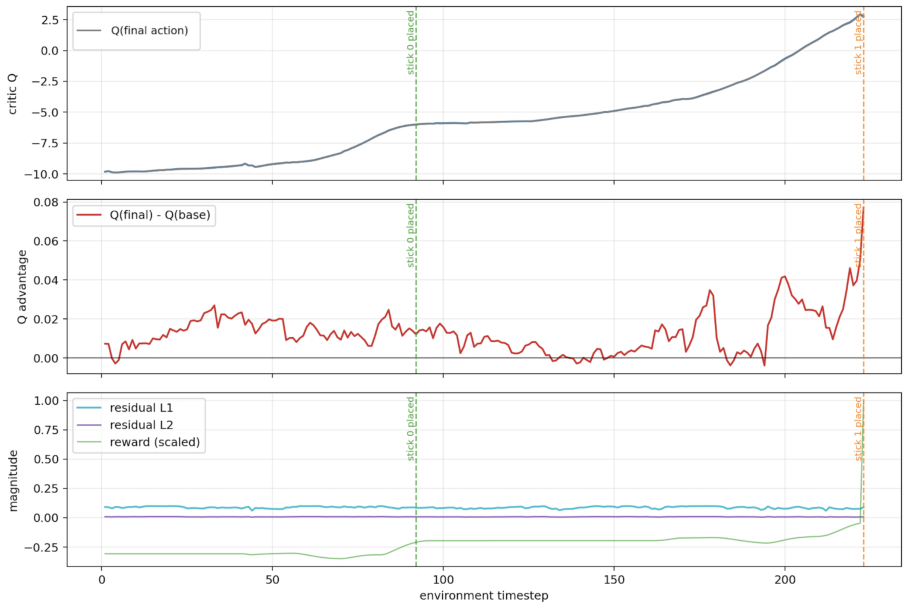

Figure 7: Diagnostics from one successful N=2 residual rollout. The critic value rises as sticks are placed, and the residual-corrected action usually has positive critic advantage over the frozen DPFM base action.

rate ratio, the proportion of offline samples in the batch, or even the planning/execution horizon of the base DPFM policy. One surprisingly fragile aspect of RRL that affects the convergence of the training is the choice of action normalization – z-normalization or min/max normalization – and subsequently, the choice of a scaling constant α. Furthermore, RRL policies proved to be slow to train, even on a simulator. However, we admit that despite using the modern ReSiFit-based algorithm, our actor and critic networks are extremely simplistic in their architecture.

Nevertheless, we believe that the up-training of the baseline behavior cloning policy through residual reinforcement learning is a highly promising, under-investigated approach to industrial robotics. We are determined to continue working in this direction. In particular, we are highly interested in merging the base policy and the residual RL policy into a single deep learning architecture using the mixture-of-experts (MoE) approach. We believe this will make the policy more expressive, training more stable, and, most importantly, remove the Achilles' heel of residual RL — the fragile linear combination of base and residual actions followed by de-normalization. Furthermore, the MoE architecture is naturally extensible with higher-level planning information, such as the action phase. However, action chunking may be non-trivial to implement within the MoE-based architecture, as the same base policy must operate in two modes: the action-chunked policy during pre-training from demonstrations, and the single-action policy during fine-tuning with RL.

8 Limitations

Our task remains a rigid-body proxy for true wire manipulation. We use state observations rather than vision, and the scripted oracle has privileged access to object and goal poses. The two-stick task is still small compared with real wire harness sorting, where the number of wires, occlusion, deformability, and contact interactions are substantially harder. The phase-active experiments also use oracle context or an expert-style phase tracker; a deployment-ready system would need to infer subtask state from observations or history.

9 Conclusion

We built and evaluated a complete imitation-to-residual-RL pipeline for wire-like robotic manipulation. Scripted demonstrations enabled strong behavior-cloning baselines. We investigated two architectures

for the base policy: a simple supervised-regression MLP-BC and a generative policy, DPFM-BC. DPFM-BC handled paired multimodal behavior better than a simple MLP, but integrating phase conditioning into the DPFM policy revealed an action-chunk context mismatch. Residual TD3 improved the selected frozen DPFM baseline from roughly 80–87% to 99% success on N=1 and from 60% to 75% success on the final N=2 task. These results suggest that residual RL is a viable refinement stage, but that the quality and architecture of the behavior-cloning base remain the cornerstone of the policy.

10 Team Contributions

- **Bautista Guerra:** Set up the initial repository, Robosuite environment, evaluation/Modal/WandB infrastructure, and scripted demonstration collection. Designed and implemented the N=2 task variants, including fixed order, balanced random order, and paired-order multimodal datasets. Adapted MLP-BC, DPFM-BC, and residual TD3 to the N=2 setting; ran the N=2 BC, multimodal, SAC, and residual-RL ablations; and produced the final analysis visualizations.
- **Andrew Yuxuan Liang:** Implemented the original MLP-BC baseline for N=1, including supervised training and closed-loop evaluation. Added phase-conditioned analysis / expert-label tooling used to study phase and active-stick context, and helped debug the N=1 residual-RL implementation.
- **Alexander Tarvo:** Implemented and debugged the DPFM-BC policy on N=1, including action chunking, inference wrappers, normalization, checkpoint loading, and DPFM training improvements. Implemented the core residual TD3 policy on top of frozen DPFM, ran the N=1 residual-RL sweeps, and documented the residual-RL training setup.

11 Changes from Proposal

The proposal planned to compare two RL fine-tuning mechanisms: an additive residual policy and LoRA-style fine-tuning of the behavior-cloning network. In the final project, we focused on the additive residual TD3 path because building the environment, demonstrations, DPFM baseline, phase/context analysis, and online residual training already formed a complete end-to-end system. We also shifted emphasis from LoRA to dataset structure and high-level context after the two-stick experiments showed that object ordering and phase inference were major failure modes.

References

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. 2025. Residual Off-Policy Reinforcement Learning for Finetuning Behavior Cloning Policies.

Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. 2024. From Imitation to Refinement: Residual Reinforcement Learning for Precise Assembly.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. 2023. Efficient Online Reinforcement Learning with Offline Data. In *International Conference on Machine Learning (ICML)*.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2024. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems*.

Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avi Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. 2019. Residual Reinforcement Learning for Robot Control. In *IEEE International Conference on Robotics and Automation*.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, and Maximilian Nickel. 2023. Flow Matching for Generative Modeling. In *International Conference on Learning Representations (ICLR)*.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. https://jmlr.org/papers/v22/20-1364.html