HINTeract: A Hierarchical Interactive Learning Framework for Robotic Manipulation

Nhi Nguyen[1], Harsh Muriki[1], Yi Lian[1]
[1]*Georgia Institute of Technology*

***Abstract*—We propose an interactive hierarchical learning framework for robot manipulation tasks that leverages minimal human supervision through on-demand hint requests. We validate our approach on a simulated furniture assembly task using Robosuite and the Sawyer robot. Low-level policies are trained via Behavioral Cloning on subtasks such as grasping, lifting, and inserting, while a high-level policy manages transitions and hint integration. We also provide more than 260 demonstrations as well as open-source the algorithms module for the simulation environment. Due to limited compute and engineering time, evaluation focused on training stability and qualitative rollout inspection rather than full-task benchmarking. Although our evaluation is preliminary, qualitative results indicate promising performance. We also discuss the lessons learned from several system-level and engineering challenges. You can access the source code and demonstration data at: https://github.com/harshmuriki/cs7648**

***Index Terms*—Interactive Learning, Hierarchical Reinforcement Learning, Robot Manipulation, Human-Robot Interaction**

I. Introduction

Robots performing complex manipulation tasks often encounter challenges interpreting subtle errors, leading to failures and inefficiencies. Traditional approaches, such as imitation learning (IL) and reinforcement learning (RL), attempt to address this problem in different ways but exhibit clear limitations. Imitation learning, for instance, generally requires extensive expert demonstrations that precisely illustrate how a task should be performed [1], [2]. These demonstrations can be expensive, time-consuming, and cognitively demanding for human teachers. Furthermore, due to humans' natural limitations in memory, attention, and cognitive load, demonstrations provided can be incomplete, inconsistent, or suboptimal, complicating the robot's learning process [3], [4].

On the other hand, reinforcement learning, which relies on trial-and-error exploration guided by reward signals, often struggles with slow convergence and inefficient exploration, especially when applied directly to complex real-world tasks [1]. Such learning processes typically require extensive interactions with the environment, which are costly and impractical in dynamic settings. Sim-to-real challenges, such as visual mismatch and dynamics discrepancies, further complicate real-world deployment and necessitate sophisticated transfer techniques like domain randomization [5]. Consequently, neither imitation learning nor reinforcement learning alone adequately addresses the challenges of real-world robot manipulation.

Interactive robot learning offers an appealing middle ground by incorporating human guidance into the robot's learning loop. In interactive methods, robots leverage timely inputs from human demonstrators (expert) during the learning process, significantly improving learning efficiency and task accuracy compared to fully autonomous trial-and-error approaches. Interactive robot learning methods can broadly be categorized into various types, including Learning from Demonstration (LfD), Learning from Feedback (LfF), and Learning from Interventions (LfI) [6]. LfD methods typically involve an expert demonstrating the desired task repeatedly. LfF methods utilize evaluative feedback, such as rewards or corrections, to shape the robot's behavior. In contrast, LfI methods engage the human instructor at critical decision points or when the robot is uncertain about how to proceed, providing targeted hints or corrective interventions.

A key aspect of interactive robot learning is the nature and timing of human-robot interactions. Human guidance can significantly accelerate learning, but overly frequent or detailed interactions can be burdensome and impractical for human instructors, leading to cognitive overload or fatigue. Conversely, insufficient interaction or delayed intervention can allow robots to accumulate errors, reducing learning effectiveness and causing frustration [6]. Balancing this trade-off, minimizing human cognitive effort while maximizing robot learning efficiency, is crucial yet challenging.

To address these limitations, we propose an interactive hierarchical learning framework (HINTeract) inspired by instructional step-by-step tasks commonly found in human manual procedures, such as IKEA furniture assembly or LEGO building. Humans naturally provide guidance in such tasks through targeted hints or brief interventions rather than exhaustive demonstrations. Inspired by the intuitive way humans guide one another during step-by-step procedures, our approach allows robots to actively solicit minimal yet precise human hints at critical decision points or when subtle task deviations occur.

Our framework decomposes complex manipulation tasks into hierarchically structured subtasks (e.g., grasping, moving, and inserting), enabling the robot to clearly monitor its own progress and detect errors early. Upon identifying deviations from expected behavior, the robot proactively requests targeted human assistance, specifying precisely which subtask is problematic. This approach substantially reduces the overhead associated with providing full task demonstrations, improves overall sample efficiency, and enhances the robot's ability to generalize skills to novel scenarios.

Ultimately, our objective is to design robots that learn more independently, reducing their reliance on exhaustive human supervision. By making interaction more intuitive and less cognitively demanding, our approach aims to enhance the practicality and scalability of interactive learning systems. Robots equipped with this capability can perform tasks with greater autonomy, making them more useful in real-world scenarios such as eldercare assistance, domestic chores, or agile manufacturing environments.

Our work makes the following contributions:

- We introduce an interactive hierarchical learning framework for robotic manipulation that minimize expert effort in the feedback.
- We curate a new dataset consisting of over 260 expert demonstration trajectories for IKEA toy table assembly using the Sawyer robot in the Robosuite simulation environment.
- We implement a modular learning pipeline that supports low-level policy training via Behavioral Cloning and manual teleoperation data collection script.
- We provide initial experimental results comparing BC, PPO, and SAC baselines, and share detailed lessons learned on data collection, reward tuning, and training stability.

Our initial implementation focuses on building the infrastructure and training low-level policies; a fully learned high-level controller and automated hint integration are planned for future work.

II. Related Work

Robot learning has long relied on paradigms such as Learning from Demonstration (LfD), Reinforcement Learning (RL), and hybrid techniques that combine aspects of both. Each paradigm presents its own set of strengths and limitations, particularly when applied to complex manipulation tasks that require nuanced understanding and adaptability.

A. Learning from Demonstration (LfD)

LfD enables robots to acquire behaviors by observing expert demonstrations, bypassing the need to design explicit reward functions [1]. This approach has shown success in many constrained tasks, such as object grasping or trajectory following. However, its reliance on high-quality, temporally aligned, and consistent expert demonstrations makes it difficult to scale. Collecting such demonstrations is time-intensive and cognitively demanding for humans, particularly in tasks with long horizons or complex sequences. Furthermore, human demonstrations can be suboptimal due to natural variability, leading to ambiguity in the robot's interpretation of the desired behavior.

B. Reinforcement Learning (RL)

RL enables autonomous learning through environmental exploration and reward feedback. Despite its promise, RL faces several challenges in real-world settings. Learning can be prohibitively slow due to the sparse or delayed nature of rewards, and the robot must often explore unsafe or inefficient behaviors before converging on successful strategies. Additionally, defining a suitable reward function for complex manipulation tasks is nontrivial and often task-specific, limiting generalizability. Algorithms like Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) have improved sample efficiency and robustness, but still require extensive training time and large amounts of interaction data [2].

C. Hybrid and Interactive Methods

Recent efforts have focused on hybrid methods that combine LfD and RL to balance autonomy with data efficiency. These approaches leverage demonstrations to bootstrap learning and subsequently fine-tune behaviors using reinforcement signals. Some methods incorporate error detection modules to trigger corrective learning phases when unexpected deviations are observed. For instance, Zhao et al. [3] propose a zero-shot fault detection framework using Bayesian Inverse Reinforcement Learning, which predicts task failures without needing direct error labels. Other works focus on improving transfer between simulated and real-world environments using domain randomization, where variations in lighting, textures, and physics increase robustness to unseen real-world settings [5]. While promising, such methods often lack fine-grained task comprehension and may misinterpret context-sensitive failures, leading to unnecessary or mistimed interventions. For example, Bobu et al. [4] argue that inducing structured representations, such as task-relevant features, is essential to scale reward learning, but most models continue to rely on unstructured or black-box reward models that fail to capture semantic nuances within complex manipulation tasks.

Another strand of work explores proactive intervention techniques in robot learning. These methods allow the robot to identify when it is likely to fail and proactively request help [1]. However, most focus on low-level anomaly detection, such as reward drop-offs or state divergence, rather than understanding task structure. As a result, they can miss more subtle errors that occur within complex subtasks (e.g., slight misalignment during an insertion), which might only be noticed at the task level.

D. Learning from Hints and Human-Robot Collaboration

Emerging interactive learning paradigms explore more structured forms of human-robot collaboration, such as learning from evaluative feedback or corrective hints. Instead of relying on full demonstrations or explicit rewards, robots are guided through targeted interventions provided at key decision points [6]. This model better reflects how humans learn in instructional contexts, for example, how a person might guide a child building LEGO by providing step-specific feedback without demonstrating the entire build. By limiting human input to brief, meaningful cues, these approaches aim to reduce cognitive burden and improve scalability.

Our work builds upon this notion by incorporating hint-driven guidance into a hierarchical task decomposition framework. Unlike previous methods that treat tasks as monolithic and reactive, our system maintains an internal representation

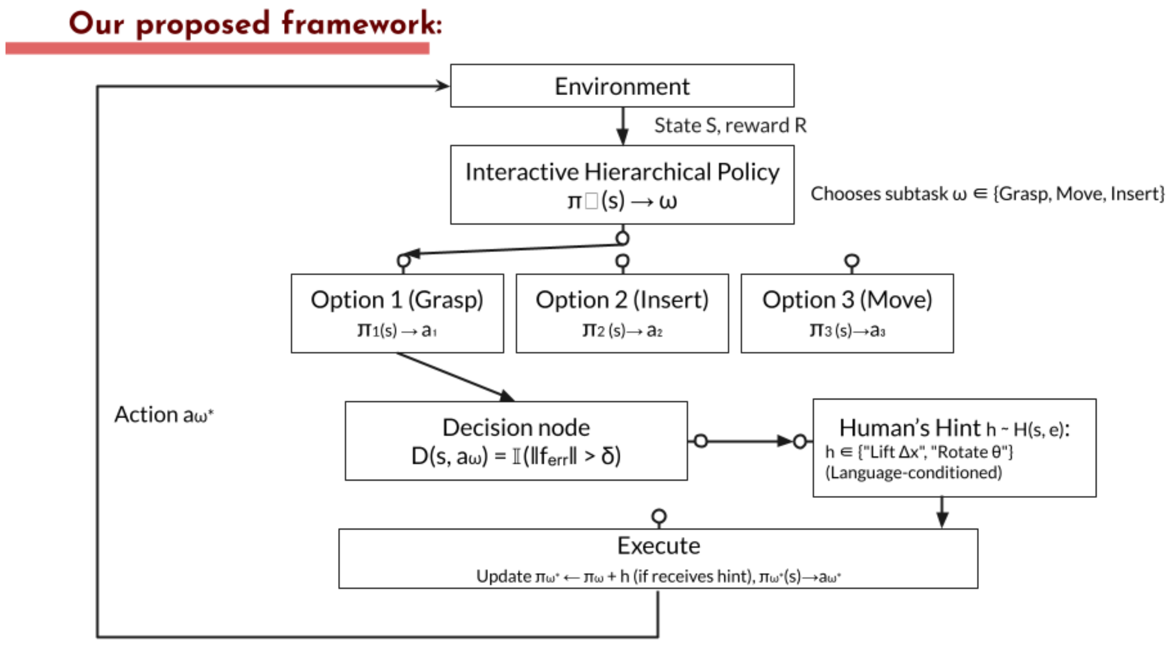

Fig. 1. Overview of our interactive hierarchical learning framework. A high-level policy monitors execution across subtasks and triggers human hints when deviations are detected.

of the task structure. This allows the robot to detect deviations at the subtask level and request help selectively and precisely, maximizing human input utility while minimizing unnecessary intervention. Our approach bridges the gap between low-level anomaly detection and high-level task understanding, paving the way for more natural, efficient, and robust interactive learning systems.

III. Methods

Our proposed framework (Fig. 1) integrates hierarchical reinforcement learning (HRL) with interactive human guidance to enable efficient and robust task learning in robotic manipulation. The method is designed to address long-horizon assembly tasks by decomposing them into interpretable subtasks, each handled by specialized low-level policies. A high-level policy coordinates subtask execution, monitors task progress, and determines when to request human assistance in the form of targeted hints. The overall system balances autonomy and human-in-the-loop control to improve sample efficiency, reduce cognitive load on human teachers, and increase generalization to novel scenarios.

A. Hierarchical Task Decomposition

We begin by representing each manipulation task as a sequence of subtasks such as *grasp*, *insert*, and *move* (Fig. 2). These subtasks reflect natural breakpoints within complex behaviors and are easier for both the robot to learn and humans to supervise. The task decomposition was manually defined based on domain knowledge but could also be extended with task segmentation techniques.

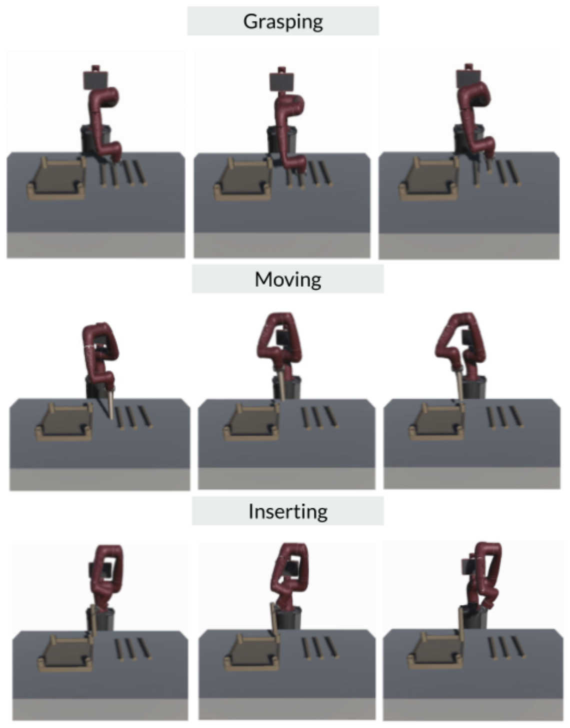

Fig. 2. Example of task decomposition for a manipulation task, broken down into subtasks such as grasp, lift, and insert.

Each subtask is associated with its own low-level controller, trained independently on expert demonstrations to perform the corresponding behavior. This modular design allows for more sample-efficient learning and more interpretable behavior, and provides explicit subtask boundaries where error detection and human intervention can be evaluated.

B. Low-Level Policy Training via Behavioral Cloning

Low-level controllers are trained using Behavioral Cloning (BC), a supervised learning technique that maps observed states to expert actions (Fig. 3). Expert trajectories were collected using keyboard teleoperation of a Sawyer robot arm within the Robosuite simulation environment. Robosuite was selected for its support of diverse manipulation tasks, MuJoCo-based physics simulation, and compatibility across platforms.

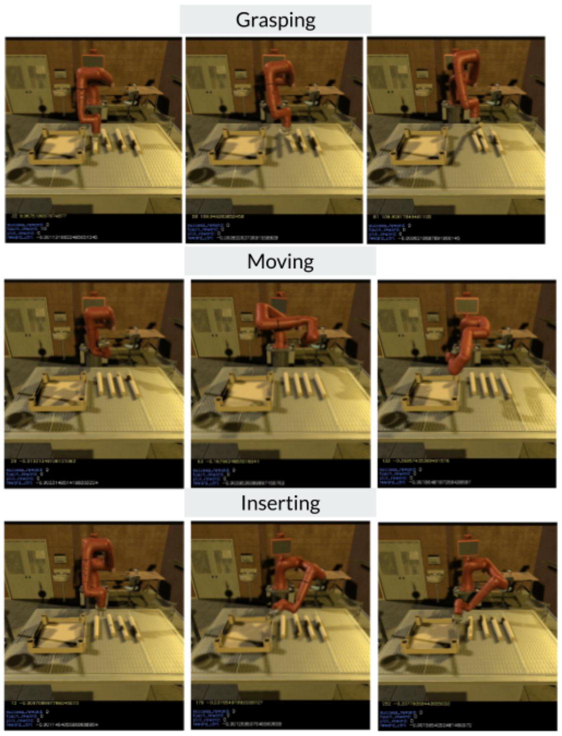

Fig. 3. Trained low-level policies via Behavioral Cloning.

Across all subtasks of the toy table assembly task, we collected over 260 demonstration trajectories, providing a robust foundation for training and evaluation. Each trajectory consists of state-action pairs representing gripper positions, joint angles, velocities, and control commands. These data were loaded through a custom `ExpertDataset` class, with optional support for low-level action traces and subsampling. The dataset was then split into training and validation sets using PyTorch's `SubsetRandomSampler`, based on a configurable validation split ratio.

Each subtask-specific policy is modeled using a feedforward `Actor` network consisting of an encoder followed by a multilayer perceptron (MLP). The encoder is either shared across modules or instantiated per subtask, depending on configuration. The MLP's architecture is configurable via the `policy_mlp_dim` parameter and uses ReLU activation function defined by `policy_activation`.

The policy outputs are treated as either deterministic vectors or, optionally, parameters of Gaussian distributions when using a stochastic variant. However, in our Behavioral Cloning setting, we use deterministic actions. The network is trained with the Adam optimizer using mean squared error (MSE) loss between the predicted and expert action vectors. Learning rate decay is scheduled via a `StepLR` scheduler, halving the learning rate every 20% of total training steps.

Training was implemented in PyTorch and accelerated on a single-machine GPU setup using a modular codebase. Input observations were normalized via a running mean and standard deviation (`self._ob_norm`) before being passed into each policy network. The training pipeline supported PyTorch-based dataloading with optional validation splits, model checkpointing, and integrated learning curve logging. Key diagnostic metrics, including actor loss, gradient norm, weight norm, and per-dimension L1 error, were recorded at each epoch to monitor convergence and identify potential training instabilities. While all experiments were run locally, the system is compatible with distributed training via MPI if needed. This setup provided a robust foundation for developing modular subtask policies later integrated into the high-level control and hint-response framework.

C. Demonstration Data Format

Each demonstration trajectory is stored as a dictionary containing synchronized data streams sampled at fixed intervals. These data keys include: `states`, `obs`, `actions`, `rewards`, `low_level_obs`, `low_level_actions`, and `connect_actions`.

- **`states`**: Each element is a dictionary representing the full simulator state at a specific timestep. Key components include:
  - `qpos`, `qvel`: Joint positions and velocities of the robot.
  - `#_part#`, `#_part#_qvel`: 7D vectors encoding the poses and velocities of individual object parts in the scene (e.g., `0_part0`, `1_part1`, etc.).

  Although the structure provides rich kinematic detail, only a subset of these fields, primarily `qpos`, `qvel`, and part poses, were used for training. Parsing these dictionaries required custom filtering logic to match network input expectations.
- **`obs`**: Each entry is an `OrderedDict` containing:
  - `object_ob`: Flattened vectors encoding object poses and orientations.
  - `robot_ob`: Internal robot state features, including joint angles, velocities, and gripper state.

  Observations are stored at one timestep greater than `actions`, consistent with the convention that each action taken at timestep t transitions the system from s_t to s_{t+1}.

- **`actions`**: An array of 8-dimensional control vectors:

$$[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$$

where the first 6 values represent motion commands (e.g., end-effector deltas), `a_6` encodes gripper action (`-1.0` = open, `1.0` = close), and `a_7` is the connect action.
- **`rewards`**: Scalar values aligned with `actions`, typically small and uniform (e.g., `-0.002`). These were not used for training, but are retained for compatibility with reinforcement learning extensions.
- **`low_level_obs` and `low_level_actions`**: High-frequency data sampled at the simulator's internal control rate. These are significantly longer than `obs`/`actions` arrays and structured as:
  - `low_level_obs`: `OrderedDict` objects mirroring `obs`, but sampled at a finer resolution.
  - `low_level_actions`: 9D control vectors, with similar semantics as `actions`, possibly including intermediate actuation commands.

  These were not used in the current training setup, but may support future applications such as contact-rich control or trajectory smoothing.
- **`connect_actions`**: Scalar values recorded at each step (typically `0.0`). The purpose is unclear, but it may encode contact flags, constraint transitions, or other simulator-specific metadata. This field was unused in training.

Understanding and integrating this data format required significant manual inspection and debugging. Shape mismatches and undocumented field structures initially led to inconsistencies in preprocessing. These issues were resolved by isolating and standardizing relevant fields, such as `qpos`, `robot_ob`, and gripper actions, into a consistent format compatible with the behavioral cloning pipeline.

D. Interactive High-Level Policy and Hint Integration

The high-level policy is implemented as a planner on top of Proximal Policy Optimization (PPO). The idea is to encourage the robot to learn task execution without full demonstrations. Our framework proposes human input in the form of structured hints, simple language instructions, integrated based on the robot's uncertainty. These hints follow an instruction–direction–value format, such as 'rotate left 15 °' or 'increase the grasp force by 10%', to ensure unambiguous interpretation. Upon receiving a hint, the corresponding subtask controller performs a brief fine-tuning phase, updating its action accordingly. A robot's uncertainty is quantified via the policy entropy

$$H\big[\pi(\cdot \mid s)\big] = -\sum_{a \in \mathcal{A}} \pi(a \mid s)\,\log\big[\pi(a \mid s)\big],$$

where a high entropy value above a threshold η triggers a hint request.

Due to time constraints and the development effort required for individual subtask policies, we replace the interactive hint mechanism with reward engineering. More precisely, we inject partial rewards at key moments to mimic human corrections. For example, once the robot successfully grasps the leg, we activate an auxiliary horizontal-distance reward:

$$\Delta r_t = \lambda\, r_{xy\_dist\_rew}(s_t, a_t) \quad \text{for } t = t_{\text{grasp}},$$

where $r_{xy\_dist\_rew}$ gives higher values as the gripper moves closer along the Y–axis toward the insertion site. Similar injections of $r_{\text{grip\_dist\_new}}$, $r_{\text{site\_up\_rew}}$, or $r_{\text{connect\_rew}}$ at their respective subtasks allow the agent to receive shaped feedback that approximates the original interactive framework. For more detailed about our rewards please check Appendix A.

E. Data Augmentation via Random Initialization

To improve generalization and task robustness, we introduced random perturbations to the initial configurations of the environment during demonstration collection and policy training. This includes small variations in the initial positions and orientations of both the robot and target objects.

Although we did not apply full domain randomization, such as altering lighting conditions, textures, or physical properties, this limited augmentation strategy still enhances the diversity of training data and helps prevent overfitting to fixed scenarios.

Policies are evaluated based on their ability to complete tasks reliably under these randomized initializations, reflecting their robustness to minor variations in starting conditions.

F. System Implementation and Environment

The entire framework is implemented in Python. Robosuite is used for environment simulation and task setup, while PyTorch is used for all neural policy models. The Sawyer robot model is chosen due to its 7-DOF control and frequent use in manipulation research. Simulation rendering is managed via Unity3D, providing high-fidelity visual feedback for debugging and demonstration capture.

The system supports real-time interaction, allowing human instructors to provide hints during execution. All human interventions and corresponding robot responses are logged for offline analysis and potential retraining of subtask controllers.

IV. Experimental Setup

To evaluate our proposed interactive hierarchical learning framework, we designed experiments in the context of a simulated furniture assembly task using Robosuite [7]. The task involved assembling a toy table by sequentially performing manipulation subtasks such as grasping table legs, aligning them with the tabletop, and inserting them correctly. This setting provides a natural testbed for long-horizon, multi-step manipulation that benefits from both structured task decomposition and human-in-the-loop correction.

A. Simulation Environment

We employed the Sawyer 7-DOF robotic arm provided in Robosuite for all experiments. The environment was configured with a toy table assembly scene, featuring a static tabletop and table legs initialized at randomized positions at the start of each episode. Gripper-based manipulation was used for all subtasks, and camera views were fixed for consistency across demonstrations and policy rollouts.

Each subtask was trained using expert demonstrations collected via keyboard teleoperation. Over 80 demonstrations were collected per subtask, with trajectories logged at 20 Hz. These demonstrations served both as training data for Behavioral Cloning (BC) and as expert references for trajectory accuracy evaluation.

B. Baselines and Benchmarks

To assess the efficacy of our method, we compared it against several commonly used baselines:

- **Behavioral Cloning (BC)**: Trained directly on full task demonstrations without subtask decomposition.
- **Soft Actor-Critic (SAC)**: An off-policy deep reinforcement learning method that optimizes a stochastic policy using entropy regularization.
- **Proximal Policy Optimization (PPO)**: A widely used on-policy algorithm that balances policy improvement with stability through clipped objective functions.

Our proposed framework, in contrast, uses a modular architecture with low-level subtask policies trained via BC and a high-level policy that governs sequencing and hint integration.

C. Evaluation Metrics and Training Observations

In principle, our framework is designed to be evaluated using three primary metrics:

- **Task Completion Rate:** The percentage of evaluation rollouts in which the robot successfully completes the full table assembly task within a fixed number of steps.
- **Intervention Frequency:** The average number of human hints requested per rollout, used to quantify the reliance on human input.
- **Trajectory Accuracy:** Measured using mean squared error (MSE) between predicted and expert actions across evaluation rollouts.

However, due to time and compute constraints, we were unable to perform full-scale quantitative evaluations across multiple rollouts. Instead, we relied on two proxies for evaluation: (1) the final episode reward averaged over late-stage training episodes, and (2) visual inspection of policy behavior during rollouts. These observations allowed us to assess overall policy stability, coherence, and success in subtasks, even in the absence of complete benchmark metrics.

Table I summarizes the final episode rewards for each algorithm, accompanied by qualitative assessments based on visual rollouts. While these results do not replace comprehensive benchmarking, they offer valuable insights into the relative learning stability and task effectiveness of each method given our time and resource constraints. For a deeper view into reward dynamics over training, Figure 4 illustrates the episode reward progression across training iterations for each algorithm. This visualization highlights not only the final performance but also the learning trends and volatility encountered during training.

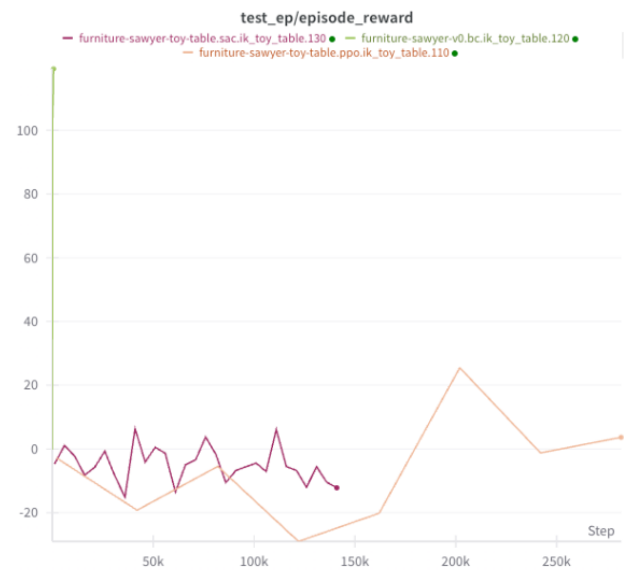

Fig. 4. Training curves showing episode reward progression over time for each algorithm (Green: BC, Red: SAC, Orange: PPO).

D. Implementation Details

All models were implemented in Python using PyTorch. Training was performed on a local workstation equipped with an NVIDIA RTX 4070 GPU. Each low-level BC model was implemented as a feedforward multilayer perceptron (MLP) with two hidden layers of 256 units each and ReLU activations. These configurations were set via the `policy_mlp_dim` and `policy_activation` parameters in the training script. Observations were normalized before being passed to the network, and model parameters were optimized using Adam with a learning rate decay schedule.

The high-level policy was implemented as a rule-based controller that detected subtask transitions and triggered human hints based on predefined thresholds or failure cues. While simple, this design allowed us to isolate the effects of hierarchical structure and targeted human interventions. Future work could incorporate a learned high-level controller for adaptive hint timing.

The experimental setup was designed to evaluate the contribution of interactive hint-based learning in combination with hierarchical decomposition, with a focus on improving sample efficiency, task completion rates, and robustness over baseline methods.

V. Results

Our initial experimental results highlight both promising trends and key challenges in implementing our interactive hierarchical learning framework. The most consistent progress was observed in the grasping subtask, where BC achieved reliable end-effector positioning. However, it sometimes failed to maintain sufficient grip force, leading to unsuccessful object lifts. This inconsistency revealed a critical weakness in low-level control policies and emphasized the need for better

TABLE I
FINAL EPISODE REWARD AND QUALITATIVE OBSERVATIONS ACROSS ALGORITHMS

Method	**Final Episode Reward**	**Qualitative Observations**
Behavioral Cloning (BC)	119.26	High and stable reward throughout training. The BC model converged quickly and maintained consistency, indicating successful imitation of expert trajectories with minimal variance.
Soft Actor-Critic (SAC)	-12.17	Reward fluctuated with no upward trend, remaining negative throughout. SAC failed to learn a stable policy in this setup, likely due to reward sparsity, poor exploration, or lack of tuning.
Proximal Policy Optimization (PPO)	3.66	Slightly positive reward with noticeable fluctuations. While PPO showed signs of improvement midway through training, the final performance plateaued at a low level, suggesting instability and underfitting.

feature engineering or alternative training strategies (e.g., using feedback-based fine-tuning or physics-informed constraints).

Visually, we observed strong performance in subtasks such as grasping and alignment for insertion, where the robot consistently executed actions resembling expert behavior. However, the move subtask remained unreliable, often resulting in suboptimal transitions. Due to time and resource constraints, we did not compute quantitative metrics such as success rate or intervention frequency. As a result, our evaluation is limited to qualitative assessments. Nonetheless, these observations offer preliminary support for the potential of structured task decomposition to facilitate targeted learning and debugging across subtasks.

Although the algorithm module is completed, the integration of the high-level policy with dynamic hint responses is still in progress, and a complete evaluation on the full table assembly task has not yet been finalized. The transition between subtasks occasionally fails due to ambiguous success conditions or insufficient sensing resolution, leading to early termination or ineffective hint responses.

We also encountered challenges related to demonstration collection. Controlling the robot via keyboard input proved cumbersome, introducing human inconsistency and noise in demonstrations. Moreover, navigating the Robosuite simulation API and tuning task-specific thresholds required substantial time and iterative debugging. These difficulties limited the number of high-quality demonstrations we were able to gather before midterm evaluation.

Despite these challenges, the modular nature of our framework has proven valuable for iterative debugging and analysis. By isolating issues to specific subtasks or decision points, we were able to more effectively diagnose model behavior, paving the way for future improvements.

VI. DISCUSSION

The development of our interactive hierarchical learning framework has underscored both the promise and complexity of designing human-in-the-loop systems for robotic manipulation. While our approach shows potential, we were ultimately limited by constraints in time, computational resources, and the complexity of building a novel system from scratch. In particular, we lacked access to high-performance compute clusters such as PACE, which restricted our ability to scale experiments and fully evaluate performance across all baselines.

A significant challenge was the design and collection of high-quality demonstrations. Acquiring demonstrations that precisely reflected our intended criteria, such as the transition between “move,” “grasp,” and “insert”, was nontrivial. Demonstration quality varied due to the people demoing using the keyboard teleoperation, which proved unintuitive and time-taking for fine-grained control. Consequently, we had to repeatedly refine the demonstration protocol, redesign action boundaries, and recollect data, leading to an iterative loop of debugging, model training, and data re-acquisition. This process consumed a substantial portion of our timeline.

We also faced considerable friction in aligning the observation and action spaces. Simulator outputs often did not match the expected input format of our training pipeline, resulting in model failures that were difficult to debug. Building the entire environment, reward structure, and high-level controller from scratch further slowed development, especially as we had no experience in building a hierarchical RL system. Due to time constraints, we were unable to fully tune reward functions or hyperparameters, a critical factor that limited overall performance and learning stability.

These challenges were most evident in our baseline comparisons. BC yielded the highest episode rewards and most stable behavior, likely because it was trained directly on expert data and did not require exploration or reward design. In contrast, PPO and SAC performed poorly. PPO exhibited unstable training dynamics and inconsistent behavior, potentially due to poor reward shaping, insufficient tuning, and limited data. SAC, while theoretically robust to noise and sample inefficiency, failed to produce meaningful behavior in our setup, likely due to the complexity of the reward space, lack of experience replay tuning, and its sensitivity to unnormalized inputs. These findings reinforce that off-policy and on-policy RL methods require substantial tuning and longer training times, which we could not support within our development constraints.

Compounding these issues was the broader problem of data scarcity. Existing public datasets for manipulation are often task-specific and lack subtask granularity or Robosuite compatibility. As a result, we were forced to collect and preprocess our own dataset, further delaying implementation and limiting the size of our evaluation set.

Despite these difficulties, the modular structure of our framework provided value. Task decomposition allowed us to isolate subtask-specific failures and debug more efficiently. Our preliminary results also suggest that integrating human hints at subtask boundaries has promise in reducing intervention burden and improving learning efficiency.

Looking forward, we aim to improve the individual subtasks, and explore automated hint generation using vision-language models (VLMs) and large language models (LLMs) to further reduce teacher burden. In parallel, we plan to revisit reward shaping and high-level policy learning, once a sufficient, valid demonstration base has been collected.

In summary, our experience highlights a broader lesson: even promising learning frameworks must grapple with practical constraints, including debugging, data collection, and system integration. Robust planning, flexible system design, and realistic timelines are essential for advancing interactive robot learning beyond research prototypes into deployable systems.

VII. Conclusion

In this work, we presented an interactive hierarchical learning framework for robotic manipulation tasks that integrates task decomposition with human-in-the-loop guidance. By structuring complex behaviors into modular subtasks and enabling robots to request targeted hints when execution deviates from expectation, our method reduces the need for exhaustive demonstrations and accelerates learning in long-horizon, high-precision tasks.

Our approach addresses several of the key limitations in conventional robot learning paradigms, namely, the inefficiencies of reinforcement learning and the burden of collecting large-scale expert demonstrations for imitation learning. Through the use of behavioral cloning at the subtask level and a high-level policy that manages progression and intervention, we improve sample efficiency while maintaining interpretability and modularity in execution.

While our initial results are preliminary, they demonstrate the viability of the core idea: human intervention, when used selectively and intelligently, can improve the robot's ability to learn complex behaviors with minimal supervision. Moreover, our experiments highlighted important lessons about system design, data quality, and the need for robust integration between simulation environments and learning pipelines.

Our current implementation relies on reward engineering heuristics due to limited time and compute resources. With more engineering capacity and access to scalable training infrastructure (e.g., PACE clusters), future iterations can incorporate fully learned high-level policies and more sophisticated interaction modeling.

Looking ahead, we envision substantial potential for this framework in real-world settings. In eldercare, for instance, robots must navigate varied and sensitive scenarios with adaptability and minimal oversight. In manufacturing, interactive systems that can learn new workflows from limited operator feedback could streamline adaptation to custom or small-batch production processes. As we continue developing this framework, we are particularly excited by the prospect of combining our method with large pretrained models (such as VLMs and LLMs) to further reduce the burden on human teachers and enhance the autonomy and usability of assistive robots.

References

[1] A. Xie, F. Tajwar, A. Sharma, and C. Finn, "When to ask for help: Proactive interventions in autonomous reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2210.10765

[2] C. Gokmen, D. Ho, and M. Khansari, "Asking for help: Failure prediction in behavioral cloning through value approximation," 2023. [Online]. Available: https://arxiv.org/abs/2302.04334

[3] H. Zhao, X. Liu, and G. Dudek, "Zero-shot fault detection for manipulators through bayesian inverse reinforcement learning," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 3582–3589.

[4] A. Bobu, M. Wiggert, C. Tomlin, and A. D. Dragan, "Inducing structure in reward learning by learning features," 2022. [Online]. Available: https://arxiv.org/abs/2201.07082

[5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017. [Online]. Available: https://arxiv.org/abs/1703.06907

[6] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, Z. Xu, D. Sadigh, A. Zeng, and A. Majumdar, "Robots that ask for help: Uncertainty alignment for large language model planners," 2023. [Online]. Available: https://arxiv.org/abs/2307.01928

[7] C. Lab, "Furniture assembly environment for robotic manipulation," https://github.com/clvrai/furniture, 2021, accessed: April 24, 2025.

Appendix A
Reward Components and Their Definitions

Reward	**Meaning**
r^{ctrl}	Penalizes excessive control effort.
$r^{\text{grip\_dist}}$	Encourages moving toward the grasp point.
$r^{\text{grip\_up}}$	Encourages vertical alignment of gripper and site.
$r^{\text{site\_dist}}$	Brings connection sites closer in space.
$r^{\text{site\_up}}$	Aligns the orientation of connection axes.
r^{xy}	Reduces horizontal distance (XY plane).
r^{z}	Reduces vertical gap (Z axis).
r^{pick}	Bonus for a successful pickup.
r^{aligned}	Bonus for correct position and orientation.
r^{connect}	Bonus for connecting at proper alignment.
r^{success}	Final reward for full part connection.