Enabling Long(er) Horizon Imitation for Manipulation Tasks by Modeling Subgoal Transitions

Shivam Jain **Sachit Sachdeva** **Rohan Paul**

Indian Institute of Technology Delhi, India

Abstract: Imitation-based policy training for long-horizon manipulation tasks involving multi-step object interactions is often susceptible to compounding action errors. Contemporary methods discover semantic subgoals embedded within the overall task, decomposing the overall task into tractable shorter-horizon goal-conditioned policy learning. However, policy deployment requires iteratively estimating *which* subgoal is being pursued and *when* it is achieved. We observe the brittleness of conventional *heuristic*-based approaches (ad hoc threshold based), particularly for long-horizon imitation, since pursuing an incorrect subgoal can lead the robot policy to experience out of distribution states. In this work, we introduce two policy architectures for modeling subgoal transitions within a policy learning loop for long-horizon tasks. The first model autoregressively predicts the likelihood of the next subgoal transition, while the second uses cross-attention (via a transformer-based architecture) and implicitly models smooth and continuous transitions. We evaluate our models on 25 simulated tasks on Franka Kitchen, 6 real-world table-top tasks and 18 simulated tasks on a new corpus (Franka-Long Horizon Tasks (LHT)) focused on tasks with rich object interactions over long episode lengths. Experimental results show significant improvements in learning efficacy, task success rates and generalization to out-of-distribution settings- extending horizon lengths for imitating manipulation tasks *from long to long(er)*. Project webpage: https://shivam89jain.github.io/lhi/

Keywords: Long Horizon Task Execution, Goal Conditioned Policy Learning, Imitation Learning, Learning from Demonstration, Robot Manipulation

1 Introduction

Training robots for long-horizon tasks remains a fundamental challenge in robot learning. These tasks often involve executing a precise sequence of actions to accomplish complex, multi-step goals. For example, assembling furniture, cooking a meal, or setting a dining table requires a robot to understand dependencies between subtasks and transition smoothly between them. Learning policies that can generalize across such long-horizon tasks is difficult due to compounding errors and the need for structured decision-making over extended sequences.

Prior work has shown that decomposing trajectories into subgoals and training goal-conditioned policies (see Fig. 1) can significantly improve learning efficacy and task success [1]. Existing policy architectures for multi-task goal-conditioned learning include MLP-based models [1, 2] and transformer-based [3] models such as the GPT-style causal transformer [4, 5, 6, 1]. These models demonstrate strong learning capabilities, but a key challenge remains during inference: given a sequence of subgoals, representing milestones of a long-horizon task, the robot must determine when a subgoal has been successfully achieved and transition to the next one. The most commonly used approach for transitioning subgoals is to define a fixed hyperparameter, ϵ, where a subgoal is considered achieved if the mean squared error (MSE) between the current state and the subgoal falls below this threshold [7, 1]. While simple and effective in some cases, this method is highly sensi-

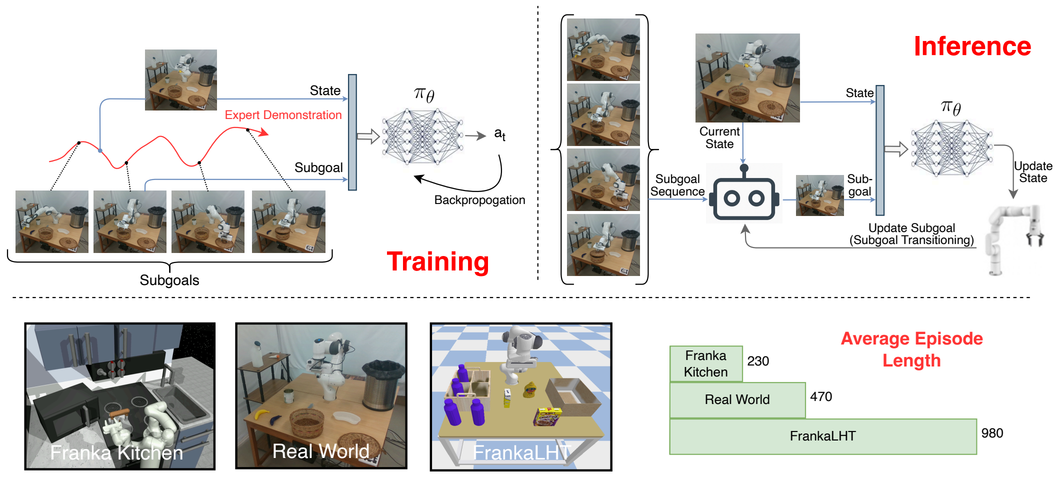


Figure 1: *(top-left)* Training goal-conditioned policies from expert demonstrations using behavioral cloning. *(top-right)* At inference time, the policy follows a sequence of subgoals to accomplish the overall task. A subgoal transition mechanism updates the subgoal based on current task progress. *(bottom)* Environments used in our experiments along with the average episode lengths (figure not to scale) of demonstration trajectories, indicating increasing task horizons.

tive to hyperparameter tuning and does not generalize well across diverse tasks with varying state distributions. This makes calculating an optimal ϵ for different environments challenging. Alternative methods for subgoal transitions include maintaining a subgoal budget [1] or avoiding this step completely by always selecting the k-th future frame as the subgoal [8]. However, these approaches require additional trajectory information during inference, such as access to the full execution path in advance, making them less practical for real-world deployment. To address these limitations, we propose a novel architecture, ST-GPT (Subgoal Transition-GPT), that allows the policy itself to predict when a subgoal has been achieved. Instead of relying on handcrafted heuristics, our model learns to infer subgoal transitions directly from experience with no additional supervision required.

However, a limitation of this entire framework is that it still relies on identifying a discrete timestamp to determine when the policy should stop conditioning on the current subgoal and transition to the next one. This can introduce instability, especially in very long-horizon tasks, where a single misidentified subgoal transition can push the robot out of distribution, leading to cascading failures. To address this, we propose a novel transformer-based architecture called SGPT (Subgoal Guided Policy Transformer). SGPT processes all subgoals simultaneously, encoding them into a structured representation. Instead of making hard transitions at discrete time steps, the current state undergoes cross-attention over the subgoal embeddings, allowing the model to dynamically adjust the weight of each subgoal in a continuous manner. This enables smooth and continuous subgoal transitions, improving policy robustness and adaptability in long-horizon tasks.

To evaluate our approach, we conduct extensive experiments in both real-world robotic setups and simulated environments. We also introduce FrankaLHT, a new simulation benchmark specifically designed for long-horizon robot learning. Compared to existing benchmarks, FrankaLHT presents more complex, multi-stage tasks that require precise long-horizon planning and execution.

Our key contributions can be summarized as follows:

1. *ST-GPT* and *SGPT*, two novel policy architectures for modeling explicit discrete (hard) and implicit continuous (soft) subgoal transitions respectively.
2. *FrankaLHT*, a challenging simulation benchmark designed for long-horizon task learning, featuring diverse scenarios across household, medical, and industrial settings.
3. Extensive experiments on multiple environments, demonstrating the scalability and adaptability of our approach.

2 Related Work

Training desired behaviors via imitation learning (IL) is a popular paradigm as an alternative to designing rewards [9]. Behavior cloning (BC) is a form of IL that casts imitation as supervised learning from observations to actions given offline demonstrations from an expert [10]. BC is susceptible to catastrophic failures when compounding errors lead the robot to states outside its training distribution [11]. The problem is more pronounced while learning skills involving multi-stage complex object interactions inherent in real-world tasks (such as assembly, packing, clearing etc.) which are the focus of this paper.

Contemporary approaches use *subgoal* decomposition of a long demonstration by observing phase shifts in visual representations [1], transitions in proprioceptive data [12] or using a video prediction model [13]. Inherently subgoals model task progress and provide a shorter-horizon learning signal during policy training. Crucially, learning subgoal conditioned policies relies on estimating when a subgoal has been achieved and when the policy should pursue the next subgoal. Prior works use *heuristic* methods such as using an ϵ-threshold [7], adjusting a subgoal budget [1] or training task-specific subgoal trackers [14, 15]. In contrast, this work learns a subgoal transition model jointly during policy learning.

Recently, transformer-based auto-regressive models have been successful as policy architectures mapping a context of robot state to next action [4, 5, 16, 6]. For robot manipulation, works such as [8, 17] achieve generalization across tasks by incorporating tokenized language annotations or goal images as context. We build on the aforementioned models and focus on incorporating image-based subgoals as well as modeling their transitions while learning a policy for a long-horizon task. Other policy architectures learn a task-specific multi-modal distribution over actions via diffusion [18], BeT [19, 20] or via Gaussian mixtures [21]. In this work, we develop a unified architecture that generalizes across multiple tasks.

This work is also related to general RL techniques that use subgoals [22] and those which relabel trajectories with subgoals [23, 24] in order to improve generalization in RL for manipulation tasks. This work confines itself to the behavior cloning setup and instead uses subgoals to improve temporal generalization during policy training. Alternative approaches use semantic knowledge of task composition to guide hierarchical skill learning via foundation models [25], querying the human [26] or by using symbolic planners [27]. While the aforementioned works use language annotations as supervision for decomposition, this work trains a policy from subgoals that capture interaction-related transitions during a long task. The subgoals lie in the visual space and may be more numerous than the subset aligned with distinct language descriptions.

3 Problem Statement

Trajectory Representation and Subgoals: A robot's interaction with its environment can be represented by its state at each time step. We define the *state* at time t as $\mathbf{S}_t = (\phi(s_t), s'_t)$, where s_t is an image captured from an external camera, offering a visual representation of the scene, and s'_t denotes the robot's joint state, encapsulating its physical configuration. ϕ denotes an image encoder that extracts a compact vector representation of the visual scene. A *trajectory* $\mathcal{T}$ is represented as a sequence of state-action pairs, formally defined as $\mathcal{T} = \{(\mathbf{S}_t, a_t)\}_{t=1}^{N}$. The ordered *sequence of states* within $\mathcal{T}$ is denoted by $\mathcal{S} = (\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_N)$. A *subgoal sequence* associated with $\mathcal{T}$ is a selected subsequence of $\mathcal{S}$, represented as $\mathcal{G} = (\mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_n)$. Each subgoal $\mathbf{G}_i$ serves as an intermediate milestone in the trajectory, with the final subgoal always aligning with the last state, i.e., $\mathbf{G}_n = \mathbf{S}_N$. For each state $\mathbf{S}_i$, the *corresponding subgoal* $\bar{\mathbf{G}}_i$ is defined as the earliest future state that belongs to the subgoal set $\mathcal{G}$. Formally, $\bar{\mathbf{G}}_i = \mathbf{S}_k$, where $k = \min\{j \mid j \geq i, \mathbf{S}_j \in \mathcal{G}\}$. This ensures that every state is mapped to the nearest succeeding subgoal in the trajectory. Consequently, the *subgoal-mapped trajectory* $\mathcal{T}'$ is defined as $\mathcal{T}' = \{(\mathbf{S}_t, a_t, \bar{\mathbf{G}}_t)\}_{t=1}^{N}$, where $\bar{\mathbf{G}}_t$ denotes the future subgoal, guiding state $\mathbf{S}_t$. We use the Universal Visual Decomposer (UVD) [1], an off-the-shelf task decomposition method for visual long-horizon manipulation, to extract a semantically meaningful subgoal sequence $\mathcal{G}$ from a given trajectory $\mathcal{T}$. Additional details on UVD can be found in Appendix A.

Policy Training: The training dataset consists of multiple trajectories and is denoted as $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^{N}$, where $\mathcal{T}_i = \{(\mathbf{S}_t, a_t)\}_{t=1}^{N_i}$. The corresponding *subgoal-mapped dataset* is given by $\mathcal{D}' = \{\mathcal{T}_i'\}_{i=1}^{N}$, where $\mathcal{T}_i' = \{(\mathbf{S}_t, a_t, \bar{\mathbf{G}}_t)\}_{t=1}^{N_i}$. The objective is to train a policy π_θ that minimizes the expected loss,

$$L = \mathbb{E}_{(\mathbf{S},a,\mathbf{G})\sim\mathcal{T}',\mathcal{T}'\sim\mathcal{D}'}\left[\mathcal{L}(a, \pi_\theta(\mathbf{S} \mid \mathbf{G}))\right]$$

where $\mathcal{L}(\cdot)$ is a predefined loss function.

Policy Evaluation: During policy evaluation, the agent is provided with a sequence of subgoals, denoted as $\{\mathbf{G}_i\}_{i=1}^{n}$. These subgoals may originate from: 1. A trajectory $\mathcal{T}_i$ in the training dataset $\mathcal{D}$, 2. A previously unseen trajectory, or 3. A semantically meaningful sequence of subgoals that is independently defined, and not derived from any trajectory decomposition. The policy begins by sampling an action a from $\pi_\theta(\mathbf{S}_1 \mid \mathbf{G}_1)$, where $\mathbf{S}_1$ represents the initial state. The selected action a is then executed by the robot, and the policy proceeds iteratively. At some time step t, once $\mathbf{G}_1$ is deemed achieved, the policy transitions to the next subgoal by updating its conditioning to $\pi_\theta(\mathbf{S}_t \mid \mathbf{G}_2)$. This process continues until the final subgoal is achieved.

4 Technical Approach

We now introduce our key contributions: ST-GPT (Subgoal Transition-GPT), a framework for modeling *explicit* subgoal transitions, and SGPT (Subgoal Guided Policy Transformer), a novel transformer-based architecture that *implicitly* models these transitions in long-horizon skill learning. We further elaborate on how these models redefine the problem formulation and enhance policy learning.

4.1 *ST-GPT: Modeling Subgoal Transitions*

ST-GPT takes as input a fixed-length history of the last k states $(\mathbf{S}_{t-k+1}, \ldots, \mathbf{S}_t)$ along with the current subgoal $\mathbf{G}$. These inputs are passed as separate tokens through a causal transformer, which employs masked self-attention to ensure each token attends only to past and present information (see Fig. 2). The transformer processes the sequence and outputs contextualized token embeddings, which are then concatenated and passed through an MLP head. The MLP produces two outputs: the joint-space action a_t, specifying the robot's intended movement, and a subgoal transition signal $\sigma_t \in \{0, 1\}$, where $\sigma_t = 1$ signifies that the current subgoal has been achieved, prompting a transition to the next subgoal, while $\sigma_t = 0$ indicates that the current subgoal remains active. Mathematically, the policy is defined as: $\pi_\theta(\mathbf{S}_t \mid \mathbf{G}) \equiv \pi_\theta(\mathbf{S}_{t-k+1}, \ldots, \mathbf{S}_t \mid \mathbf{G}) = (a_t \circ \sigma_t)$. Given a *subgoal-mapped* trajectory $\mathcal{T} = (\mathbf{S}_t, a_t, \bar{\mathbf{G}}_t)$, we define a *transition-augmented subgoal-mapped* trajectory as: $\mathcal{T}' = (\mathbf{S}_t, a_t \circ \sigma_t, \bar{\mathbf{G}}_t)$, where $\sigma_t = \mathbb{I}\{\mathbf{S}_t = \bar{\mathbf{G}}_t\}$ is an indicator function that returns 1 when the current state matches the corresponding subgoal, signifying its successful completion. The corresponding training dataset $\mathcal{D}' = \{\mathcal{T}_i'\}_{i=1}^{N}$ is a collection of such *transition-augmented* trajectories. Extending our earlier formulation, the training loss for *ST-GPT* is defined as:

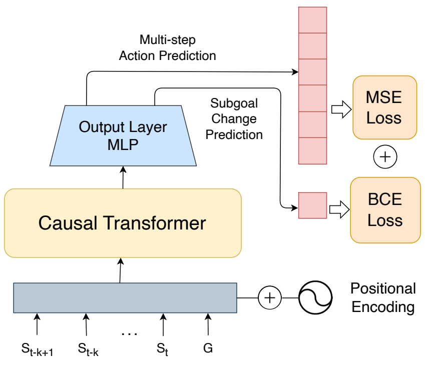


Figure 2: *ST-GPT Architecture*: The dual-headed output enables the model to determine when to transition between subgoals while executing precise motor commands.

$$L = \mathbb{E}_{(\mathbf{S},a\circ\sigma,\mathbf{G})\sim\mathcal{T}',\mathcal{T}'\sim\mathcal{D}'}\left[\mathcal{L}(a \circ \sigma, \pi_\theta(\mathbf{S} \mid \mathbf{G}))\right]$$

where $\mathcal{L}(a \circ \sigma, \hat{a} \circ \hat{\sigma}) = \mathcal{L}_1(a, \hat{a}) + w\mathcal{L}_2(\sigma, \hat{\sigma})$ is a weighted loss combination. In our framework, we use $\mathcal{L}_1$ as mean squared error (MSE) to regress the joint actions and $\mathcal{L}_2$ as binary cross-entropy

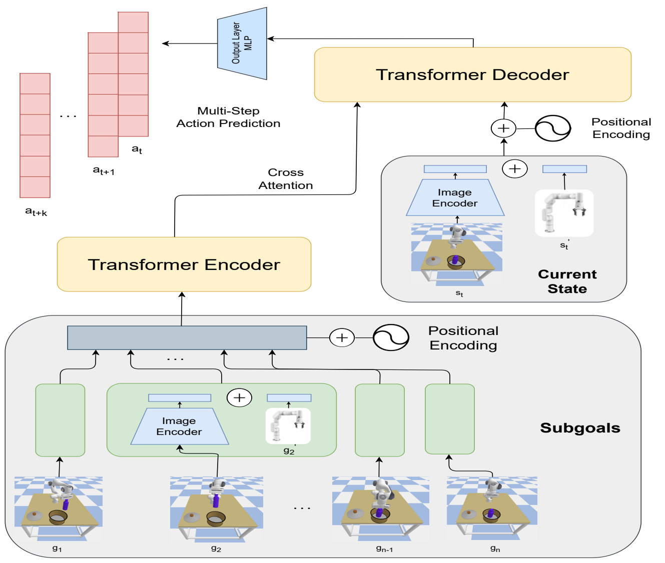


Figure 3: *SGPT Architecture*: Cross-attention over subgoals allows the model to capture subgoal dependencies, enabling fluid subgoal transitions and seamless execution of long-horizon tasks.

(BCE) to supervise the subgoal transitions. At inference time, the agent is provided with a sequence of subgoals $\{\mathbf{G}_i\}_{i=1}^{n}$ that guides the robot through a complex-long task. The key idea is that, rather than relying on manually defined thresholds or heuristic rules to determine subgoal transitions, the policy learns to predict them directly via the transition signal σ_t, jointly alongside the action a_t. In practice, instead of predicting a single action at each step, we extend the model to predict a sequence of l future joint-space actions, $(\{a_t, a_{t+1}, \ldots, a_{t+l-1}\} \circ \sigma_t)$ and apply action chunking [16] along with temporal averaging to ensure smoother behavior and improved stability. Additional details, including implementation specifics and hyperparameters, are provided in Appendix B.2.

4.2 *SGPT: Smoothening Subgoal Transitions*

While *ST-GPT* effectively models subgoal transitions, it still operates under the limiting assumption that transitions occur at discrete time steps. Identifying the exact moment of when to transition to the next subgoal is often ambiguous—even for humans. In such architectures, this reliance on hard, discrete transition points limits scalability to very long-horizon tasks, where such decisions become increasingly uncertain and can lead to cascading errors. Marking a subgoal as completed too early can lead to premature transitions and compounding errors, while delaying the transition can cause inefficient behavior or stagnation. To address these challenges, we propose *SGPT*, an architecture that enables smooth and implicit subgoal transitioning, allowing the policy to gradually shift its focus between subgoals without requiring explicit transition signals. *SGPT* takes as input the current state $\mathbf{S}_t$ along with the complete subgoal sequence $\mathcal{G} = (\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_n)$ (see Fig. 3), enabling it to contextualize its decisions over the entire task plan. To retain the temporal order of the subgoals, positional encodings are added to the sequence before passing it through a transformer encoder with multi-head self-attention, producing contextualized subgoal embeddings. The current state token $\mathbf{S}_t$ then attends to these embeddings via a cross-attention mechanism, effectively learning a soft alignment over the subgoal sequence. This process yields a weighted mixture over subgoals that reflects their relevance at the current time step. The resulting context vector is then passed through an MLP head to produce the predicted action a_t. Mathematically, the policy is defined as: $\pi_\theta(\mathbf{S}_t \mid \mathcal{G}) \equiv \pi_\theta(\mathbf{S}_t \mid \mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_n) = a_t$. Given a dataset $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^{N}$, where each trajectory $\mathcal{T}_i = \{(\mathbf{S}_t, a_t)\}_{t=1}^{N_i}$, the training loss for *SGPT* is defined as:

$$L = \mathbb{E}_{(\mathbf{S},a)\sim\mathcal{T},\mathcal{T}\sim\mathcal{D}}\left[\mathcal{L}(a, \pi_\theta(\mathbf{S} \mid \mathcal{G}))\right]$$

where $\mathcal{G}$ is the *subgoal sequence* associated with $\mathcal{T}$. In our implementation, we use mean squared error (MSE) as the loss function. To encourage more stable and smooth behavior, we also incorporate action chunking [16]—by predicting multi-step action sequences instead of single-step actions—as introduced in the previous section. While it is possible to augment *SGPT* with a history of past states as additional input, we find through ablations that this leads to suboptimal performance (see Appendix B.2). Thus, we keep the architecture simple by conditioning only on the current state. For more architectural details, please refer to Appendix B.3.

5 Experimental Setup

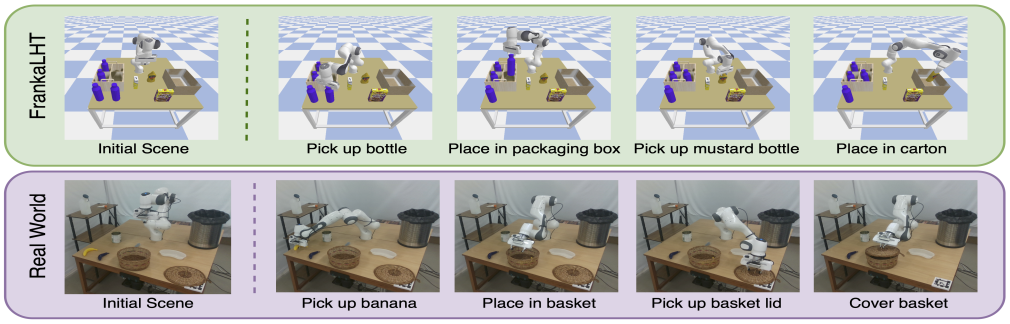


Figure 4: Sample execution trajectories from simulation(FrankaLHT) and real-world environments. These examples demonstrate the compositional and long-horizon nature of the tasks used in our experiments.

Baselines: We evaluate three state-of-the-art models for goal-conditioned policy learning, listed in increasing order of their modeling power: *MLP*, *GPT*, and *BAKU*. The ***MLP*** [1] model is a standard goal-conditioned policy that takes the current state $\mathbf{S}_t$ and goal $\mathbf{G}$, concatenates them, and passes the result through a multi-layer perceptron to predict joint-space action a_t. The ***GPT*** [1] model is a history-aware, goal-conditioned causal transformer, similar in design to the *ST-GPT* model described earlier, except that it only outputs actions and does not predict subgoal transition signals. In both these models, subgoal transitions are triggered using a fixed threshold ϵ, where the current subgoal is considered achieved when the distance $\|\mathbf{S}_t - \mathbf{G}\|_2 < \epsilon$. The ***BAKU*** [8] model takes as input multiple camera views, the robot's joint state, and a trainable action token to predict the next action. Unlike the other models, *BAKU* does not decompose the trajectory into subgoals and instead conditions on the final frame alone as the ultimate goal. It serves as a strong baseline, outperforming prior state-of-the-art methods on long-horizon manipulation tasks. All three baseline models use action chunking and predict multi-step actions. Further architectural details of these baselines are provided in Appendix B.

Evaluation Metrics & Generalization: We train multitask policies across different benchmarking environments, each consisting of n different long-horizon tasks and k expert demonstrations per task, resulting in a total of $n \times k$ trajectories. Each task's k demonstrations are collected with multiplicative Gaussian noise applied to the robot's initial state and object positions, along with inherent variability in task progress. We randomly sample half of the trajectories for each task ($k/2$) to form the training set, denoted as $\mathcal{D}_{\text{train}}$, resulting in a total of $n \times \frac{k}{2}$ training demonstrations. The remaining $k/2$ trajectories per task form the evaluation set, $\mathcal{D}_{\text{eval}}$, with the complete dataset given by $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{eval}}$. At inference time, we extract subgoals (final goal for *BAKU*) from each trajectory in $\mathcal{D}$ using UVD [1]. When the policy is evaluated on subgoals extracted from the trajectories in $\mathcal{D}_{\text{train}}$, we refer to the results as *in-distribution* (InD) performance. Conversely, evaluations on subgoals extracted from the trajectories in $\mathcal{D}_{\text{eval}}$ are referred to as *out-of-distribution* (OoD) performance. Each task is long-horizon and composed of multiple subtasks. We report two key metrics: *success rate*, defined as the fraction of roll outs in which the entire task is completed, and *completion rate*, defined as the average fraction of sub-tasks completed within a long-horizon task by the policy across different roll outs.

Our experiments are designed to answer the following questions:

- *Does the ST-GPT policy provide more effective subgoal transitions compared to the standard GPT policy that relies on a fixed ϵ threshold as a heuristic for subgoal transitions?*
- *Do continuous and smooth subgoal transitions (SGPT) outperform discrete subgoal transitions (ST-GPT)?*
- *How well do these policies scale with increasing task horizon lengths, and what is the breakpoint where their performance begins to degrade?*

6 Results

A) Simulation Experiments

Franka Kitchen Environment: We conduct our experiments in the Franka Kitchen environment, a widely used benchmark for evaluating long-horizon manipulation tasks. Each task in this environment is a sequence of 4 high-level subtasks, such as opening cabinets, turning on burners, flipping a light switch, opening the microwave, etc. The dataset $\mathcal{D}$ used in our experiments is a curated subset of the original Franka Kitchen dataset introduced in [28], consisting of 25 such tasks, with 10 demonstration trajectories per task, and an average episode length of $\sim$250 steps. For each task, 5 trajectories are randomly sampled to form the training set $\mathcal{D}_{\text{train}}$, while the remaining 5 are assigned to $\mathcal{D}_{\text{eval}}$, as described earlier. Table 1 presents a comparison between ST-GPT and the standard GPT model. We perform a grid search over different values of the hyperparameter ϵ, and show that no choice of threshold yields performance that surpasses our model. Table 2 presents a detailed comparison of all baselines, including ST-GPT and our strongest model, SGPT. The results indicate that while ST-GPT performs well on in-distribution (InD) tasks and BAKU excels on out-of-distribution (OoD) tasks, SGPT consistently outperforms all the models across both InD and OoD settings. Figure 5 in appendix provides a visualization of continuous subgoal transitioning in the SGPT architecture.

Architecture	Subgoal Transition Threshold	InD		OoD	
		Success	Completion	Success	Completion
GPT	0.003	0.320	0.696	0.040	0.456
GPT	0.001	0.480	0.770	0.040	0.369
GPT	0.0008	0.504	0.792	0.008	0.292
GPT	0.0005	0.432	0.754	0.008	0.260
GPT	0.0001	0.312	0.670	0.000	0.171
ST-GPT (**Ours**)	N.A.	**0.928**	**0.978**	**0.089**	**0.488**

Table 1: Comparison between ST-GPT and GPT models with varying subgoal transition thresholds in the Franka Kitchen environment. As the threshold ϵ decreases, OoD performance degrades due to the difficulty of achieving precise subgoal transitions in unseen tasks. InD performance peaks at an intermediate value of ϵ, indicating an optimal trade-off. Nonetheless, none of the GPT variants surpasses the performance of ST-GPT.

FrankaLHT Environment: The results from the Franka Kitchen environment show that while SGPT outperforms all other models, the baselines also perform reasonably well. We attribute this to the relatively low complexity of tasks in the Franka Kitchen environment and the increased capabilities of modern policies, which allow us to extend the notion of long-horizon tasks well beyond 250 time steps. However, no publicly available benchmarking environments focus on complex, fine-grained, long-horizon manipulation that demands sequential decision-making across multiple interconnected subtasks. To address this gap, we introduce FrankaLHT, a new benchmark specifically designed for evaluating long-horizon robot learning in complex, multi-stage tasks. Built on the PyBullet [29] physics engine, FrankaLHT comes with 180 expert trajectories spanning 18 diverse tasks across three distinct scenarios, with an average episode length of $\sim$1000 steps. With

10 trajectories per task, we generate our $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{eval}}$ datasets in the same manner as described earlier for the Franka Kitchen environment. Table 2 presents a detailed comparison of all the models across both these simulation environments. We observe that all models, including ST-GPT, perform poorly in the FrankaLHT environment. In contrast, SGPT significantly outperforms these models, achieving performance comparable to that in the Franka Kitchen environment, with only a slight drop in OoD performance. These results highlight the robustness and strong generalization capabilities of SGPT's continuous subgoal transitioning mechanism in addressing the increased complexity and longer temporal horizons posed by the FrankaLHT environment. Additional details about the FrankaLHT benchmark—including task descriptions and a comparison between ST-GPT and GPT in this setting—are provided in Appendix C.

Architecture	**Franka Kitchen**				**FrankaLHT**			
	InD		**OoD**		**InD**		**OoD**	
	S	**C**	**S**	**C**	**S**	**C**	**S**	**C**
MLP	0.040	0.452	0.008	0.254	0.000	0.100	0.000	0.092
GPT	0.504	0.792	0.008	0.292	0.000	0.175	0.000	0.067
BAKU	0.488	0.790	0.258	0.583	0.011	0.083	0.000	0.067
ST-GPT (**Ours**)	**0.928**	0.978	0.089	0.488	0.000	0.242	0.000	0.181
SGPT (**Ours**)	**0.928**	**0.982**	**0.266**	**0.639**	**0.922**	**0.969**	**0.222**	**0.381**

Table 2: Performance comparison of all architectures across both the simulation environments. For MLP and GPT, we report results using the subgoal transition threshold that yields the best InD performance. While most models perform well in the simpler Franka Kitchen setting, their performance drops significantly in the more challenging FrankaLHT environment. SGPT consistently outperforms all baselines across both settings (S: Success rate, C: Completion rate).

B) Real World Experiments

We perform our real-world experiments on a Franka Panda robot across 6 diverse table-top manipulation tasks. Figure 4 showcases the experimental setup with accompanying snapshots of a sample task. We collect 14 teleoperated demonstration trajectories per task, with an average episode length of $\sim$500 steps. These trajectories are then uniformly split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{eval}}$ in the same manner as described earlier. Table 3 presents a performance comparison of all the models in the real-world setup. We observe similar trends as in the FrankaLHT environment, with most models struggling to perform well, whereas SGPT consistently achieves superior performance over all baselines. Figure 11 in appendix provides an overview of all the real-world tasks.

Architecture	**InD**		**OoD**	
	S	**C**	**S**	**C**
MLP	0.000	0.071	0.000	0.024
GPT	0.000	0.119	0.000	0.077
BAKU	0.000	0.060	0.000	0.048
ST-GPT (**Ours**)	0.000	0.202	0.000	0.137
SGPT (**Ours**)	**0.524**	**0.810**	**0.167**	**0.411**

Table 3: Real World Experimental Results (S: Success rate, C: Completion rate)

7 Conclusion

This paper addresses the problem of determining - *which subgoal to pursue and when to transition to another* during visual BC-based imitation learning for long-horizon tasks. We introduced two policy architectures: one that *explicitly* predicts subgoal attainment and another that *implicitly* models subgoal transitions via cross-attention, both aimed at enabling policies to better navigate the long-horizon task structure. Experiments were performed on simulation environments and on a real Franka Emika Robot Manipulator over temporally extended tasks involving multi-step object interactions. Results demonstrate significant gains over state-of-the art multi-task, goal-conditioned policy learners, particularly in the long-horizon task setting, indicating the benefit of modeling subgoal transitions within the policy learning loop.

8 Limitations

A key limitation of the current framework is scalability to imitating very long sequences (e.g., packing large number of bottles in a container involving periodic object placements at slightly different locations). Imitating such tasks requires more semantic decomposition instead of attending over all constituent subgoals in the demonstration. A possible direction is to align subgoals with language to discover more pronounced compositional structure embedded in the demonstration. Finally, the current framework assumes access to subgoals from a demonstration trajectory during policy execution. Incorporating the ability to induce subgoals from related tasks can take a step in the direction of more generic goal-conditioned policies capable of performing tasks even in the absence of demonstrations.

Acknowledgments

Authors thank Mr. Nadir Mohd. for setting up the physical testbed and creating the simulation models. We thank Dr. Shubhangi Nema and Mr. Sujeet Lahane for providing training with the robot manipulator and associated software interfaces. Rohan Paul thanks Dr. Piyush Chanana, Prof. P. V. M. Rao, and the ICMR-IIT Delhi National Center for Assistive Health Technologies (NCAHT) for supporting this research activity under frontier AT research and development. Rohan Paul also thanks the Industrial Research & Development Unit, IIT Delhi, for the Equipment Grant. We are grateful to Prof. Parag Singla for providing insightful feedback on this research during project presentations. We also thank Dr. Michael Noseworthy for sharing feedback on this work. The authors are grateful to Mr. Namasivayam K for supporting the computing infrastructure and student teaching activities. Finally, we thank Mr. Shailendra Negi and Mrs. Sunita Negi for their administrative support.

References

[1] Z. Zhang, Y. Li, O. Bastani, A. Gupta, D. Jayaraman, Y. J. Ma, and L. Weihs. Universal visual decomposer: Long-horizon manipulation made easy, 2023. URL `https://arxiv.org/abs/2310.08581`.

[2] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation, 2022. URL `https://arxiv.org/abs/2203.12601`.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1:8, 2019.

[5] R. Bonatti, S. Vemprala, S. Ma, F. Frujeri, S. Chen, and A. Kapoor. Pact: Perception-action causal transformer for autoregressive robotics pre-training, 2022. URL `https://arxiv.org/abs/2209.11133`.

[6] Z. Hou, T. Zhang, Y. Xiong, H. Pu, C. Zhao, R. Tong, Y. Qiao, J. Dai, and Y. Chen. Diffusion transformer policy, 2024. URL `https://arxiv.org/abs/2410.15959`.

[7] Y. Lee, E. S. Hu, Z. Yang, and J. J. Lim. To follow or not to follow: Selective imitation learning from observations, 2019. URL `https://arxiv.org/abs/1912.07670`.

[8] S. Haldar, Z. Peng, and L. Pinto. Baku: An efficient transformer for multi-task policy learning, 2024. URL `https://arxiv.org/abs/2406.07539`.

[9] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[10] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[11] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[12] L. X. Shi, A. Sharma, T. Z. Zhao, and C. Finn. Waypoint-based imitation learning for robotic manipulation. *arXiv preprint arXiv:2307.14326*, 2023.

[13] S. Nair and C. Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *arXiv preprint arXiv:1909.05829*, 2019.

[14] A. Mohtasib, E. A. Ghalamzan, N. Bellotto, and H. Cuayáhuitl. Neural task success classifiers for robotic manipulation from few real demonstrations. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[15] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 1503–1510. IEEE, 2015.

[16] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL `https://arxiv.org/abs/2304.13705`.

[17] H. Bharadhwaj, J. Vakil, M. Sharma, A. Gupta, S. Tulsiani, and V. Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking, 2023.

[18] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[19] T. Agrawal, D. Agarwal, M. Balazia, N. Sinha, and F. Bremond. Multimodal personality recognition using cross-attention transformer and behaviour encoding. *arXiv preprint arXiv:2112.12180*, 2021.

[20] S. Lee, Y. Wang, H. Etukuru, H. J. Kim, N. M. M. Shafiullah, and L. Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

[21] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. Pmlr, 2020.

[22] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

[23] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

[24] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[25] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.

[26] M. Parakh, A. Fong, A. Simeonov, T. Chen, A. Gupta, and P. Agrawal. Lifelong robot learning with human assisted language planners. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 523–529. IEEE, 2024.

[27] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3795–3802. IEEE, 2018.

[28] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning, 2019. URL https://arxiv.org/abs/1910.11956.

[29] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[30] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

[31] A. F. Agarap. Deep learning using rectified linear units (relu), 2019. URL https://arxiv.org/abs/1803.08375.

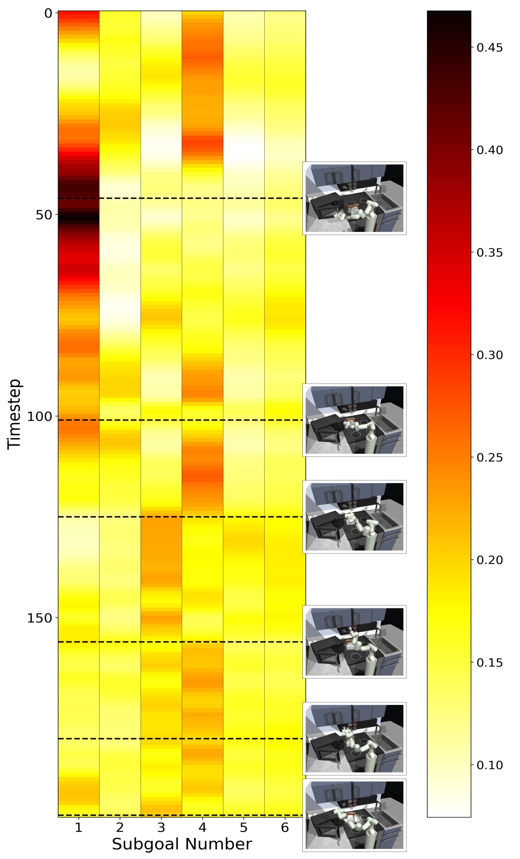
[32] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

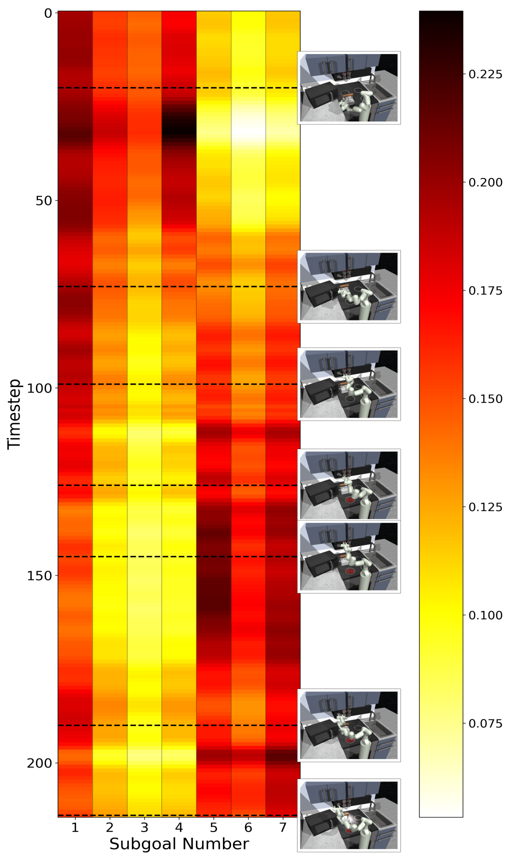
[34] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

[35] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023. URL https://arxiv.org/abs/2212.06817.

Appendix



(a) *Task sequence:* Microwave, Kettle, Top Left Burner, Light Switch
Decoder Layer: 3 *Attention Head:* 1
The first vertical bar of the heatmap illustrates how the model's attention is distributed over Subgoal 1 across time. We observe that the attention on Subgoal 1 gradually increases from the start and peaks around timestep 50. Notably, this peak aligns closely with the time at which the first subgoal occurs in the expert demonstration. This illustrates how the model does not make any hard or discrete subgoal transitions, but instead shifts its focus smoothly through gradually changing attention weights.

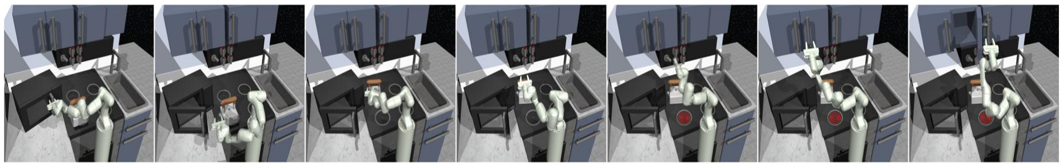


(b) *Task sequence:* Kettle, Bottom Left Burner ,Top Left Burner, Light Switch
Decoder Layer: 1 *Attention Head:* 3
We observe similar results in this heatmap as in (a), with Subgoal 5 receiving peak attention near timestep 150. Interestingly, we also notice a strong early peak for Subgoal 4, even though it occurs much later in the time horizon. This suggests that the model is attending to future subgoals early on, potentially to establish a longer-term understanding of the task. Such behavior indicates that different transformer layers and attention heads might be capturing varying levels of temporal abstraction and subgoal dependencies.

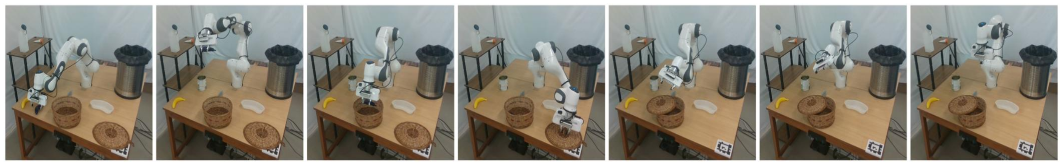
Figure 5: Cross-attention heatmaps from the SGPT policy, visualizing continuous subgoal transitioning across long-horizon tasks, in the Franka Kitchen environment. Each heatmap is generated during policy execution, where the policy is conditioned on a sequence of subgoals (shown as images to the right of the heatmap, arranged from top to bottom). At each timestep, the current state attends to these subgoals via a cross-attention mechanism to compute the action. The heatmaps display the attention weights assigned to each subgoal (x-axis) over time (y-axis, running from top to bottom). The sequence of subgoals shown here is extracted from the decomposition of some trajectory using UVD. The horizontal dotted lines indicate the frame numbers at which these subgoals occur in the video demonstration.

A Universal Visual Decomposer

We use the Universal Visual Decomposer (UVD) [1] to extract subgoals from demonstration videos, where each subgoal corresponds to a meaningful subtask within the long-horizon activity. Mathematically, given a video demonstration of a long-horizon robotic task $\mathcal{V} = (f_1, f_2, \ldots, f_N)$, where f_i denotes the i-th frame, the Universal Visual Decomposer produces a sequence of subgoals $\texttt{UVD}(\mathcal{V}) \mapsto (g_1, g_2, \ldots, g_n)$, where each g_i corresponds to a selected frame from $\mathcal{V}$, i.e., $g_i = f_j$ for some $j \in \{1, \ldots, N\}$. UVD identifies these subgoals by detecting phase shifts in the visual embedding space. These subgoals serve as intuitive checkpoints (see Fig. 6), guiding the robot through the task in a structured and interpretable manner.



(a) The robot sequentially opens the microwave, places the kettle on the top-left burner, turns the burner knob, and opens the hinge cabinet.



(b) The robot picks up a brinjal from the table, places it in the basket, and then covers the basket with its lid.

Figure 6: Examples of subgoal extraction using UVD across (a) the Franka Kitchen simulation environment, and (b) our real-world experimental setup.

B Network Architecture and Training Details

The input proprioceptive state s'_t in both the FrankaLHT and real-world setup is an 8-dimensional vector consisting of the 7 joint angles of the Franka arm and the gripper position. The action space is also 8-dimensional, where the first 7 values correspond to joint commands and the last value controls the gripper position, applied symmetrically to both fingers. The other half of the robot's state s_t comes from the embedding of an external camera image. Each input image is center-cropped to a size of 224×224 pixels and passed through the VIP encoder [30], which generates a 1024-dimensional image encoding. Table 4 presents the hyperparameters for the MLP, GPT, and SGPT policy architectures. To ensure a fair comparison across all policies, each was designed to have approximately 25M trainable parameters and trained for 1000 epochs. The only exception is the MLP policy, which has 3M parameters, as deeper or wider configurations resulted in unstable training and poor performance. To compensate for the smaller model size, we increased its training to 10,000 epochs.

B.1 BAKU

The hyperparameters for BAKU are adopted directly from the configuration provided in their original work [8]. The BAKU paper performs an extensive ablation study to justify various architectural choices in their design. We use the configuration that employs a transformer-based observation trunk, an MLP action head, no observation history, and action chunking. The policy is conditioned on the final goal image. As discussed in their paper, this configuration was shown to yield state-of-the-art results when compared to prior baselines such as MT-ACT [17] and RT-1 [35]. Notably, although BAKU does not explicitly utilize subgoals, its policy remains highly effective on long-horizon tasks.

Hyperparameter	Value
Hidden Layers	[1024, 512, 256]
Activation	ReLU [31]
Normalization	Batch Norm [32]
Epochs	10000
Batch Size	512
Learning Rate	3e-4
Optimizer	Adam [33]
Scheduler	Cosine Annealing
Action chunk length	25
Trainable Parameters	3M

(a) MLP

Hyperparameter	Value
Observation History	1
Attention Heads	4
Feedforward Dim.	1024
Layers	4
Activation	ReLU [31]
Normalization	Layer Norm [34]
Epochs	1000
Batch Size	512
Learning Rate	3e-4
Optimizer	Adam [33]
Scheduler	Cosine Annealing
Action chunk length	10
Trainable Parameters	25.7M

(b) GPT

Hyperparameter	Value
Encoder Layers	1
Decoder Layers	3
Attention Heads	4
Feedforward Dim.	1024
Dropout	0.1
Activation	ReLU [31]
Normalization	Layer Norm [34]
Epochs	1000
Batch Size	512
Learning Rate	3e-4
Optimizer	Adam [33]
Scheduler	Cosine Annealing
Action chunk length	10
Trainable Parameters	25.6M

(c) SGPT

Table 4: Hyperparameters for MLP, GPT, and SGPT policy architectures.

B.2 ST-GPT

The hyperparameters of the ST-GPT policy largely follow those of the GPT architecture, with a few key additions specific to our subgoal prediction framework. Since subgoal transitions are significantly sparser than non-subgoal transitions in long-horizon tasks—owing to subgoals representing high-level subtask boundaries rather than every step of the trajectory—we address this class imbalance using a weighted Binary Cross Entropy (BCE) loss. Specifically, we assign a `pos_weight` of 6 to place greater emphasis on positive samples. This BCE loss is combined with the Mean Squared Error (MSE) loss used for action prediction, with a weighting ratio of 100:1 (MSE:BCE) to balance the two objectives. We find, however, that the training process is not particularly sensitive to the exact values of both these hyperparameters. Table 4 (b) shows that the observation history length is set to 1, meaning only the current state and the subgoal are provided as input. We observe that including 2-3 past states yields no performance improvements, and increasing the history beyond this actually degrades performance, likely due to increased distribution shifts. We observe a similar trend with the SGPT architecture as well. As a result, we keep both these architectures simple by using a history length of 1. Similar findings have been reported in prior work [8]. Consequently, the only advantage of the GPT policy over the MLP is that in GPT, the current state and subgoals are

Architecture	Franka Kitchen	FrankaLHT	Real-World Setup
MLP	10.9	25.5	5.1
GPT	4.8	10.7	1.4
BAKU	7.9	21.6	2.6
ST-GPT	4.9	10.8	1.4
SGPT	20.1	115.8	23.4

Table 5: Training time (in hours) required for each policy architecture across different experimental setups.

passed as separate tokens, enabling the attention mechanism to model their interaction, whereas in the MLP, they are concatenated and processed simply through a neural network.

B.3 SGPT

Figure 5 shows a visualization of continuous subgoal transitions observed during the execution of the SGPT policy in the Franka Kitchen environment. The hyperparameters used for training the SGPT policy are listed in Table 4(c).

B.4 Computational Requirements

We trained all our policies on an NVIDIA A40 GPU. Table 5 reports the training time for each policy across both simulation and real-world environments. Despite having only 3M parameters, the MLP model takes longer to train due to running 10 times more epochs. We also conducted ablations with MLP variants—altering the number of parameters and epochs—but the model inherently lacks the capacity to learn complex patterns required for long-horizon manipulation. Among all policies, SGPT incurs the highest training cost, even though all models (except MLP) are trained with $\sim$25M parameters and for 1000 epochs. This is because each update step in SGPT is computationally more intensive. Unlike GPT and BAKU, which use decoder-only and encoder-only transformer architectures respectively, SGPT employs both an encoder and a decoder. Thus, while SGPT achieves the best performance across all our experiments, it comes with a significantly higher training cost.

C FrankaLHT Simulation Environment

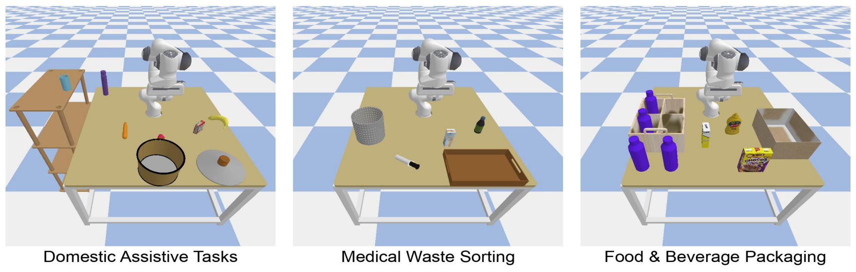


Figure 7: Overview of the three diverse scenes used for task execution in the FrankaLHT benchmark.

The FrankaLHT (Long-Horizon Tasks) environment is built on the PyBullet physics engine and provides a challenging benchmark for evaluating policies on temporally extended manipulation tasks. The environment includes 180 demonstration trajectories collected via keyboard teleoperation, with the Franka Panda robot operating at a control frequency of 50 Hz. Figures 8–10 illustrates three diverse task categories represented in the FrankaLHT suite:

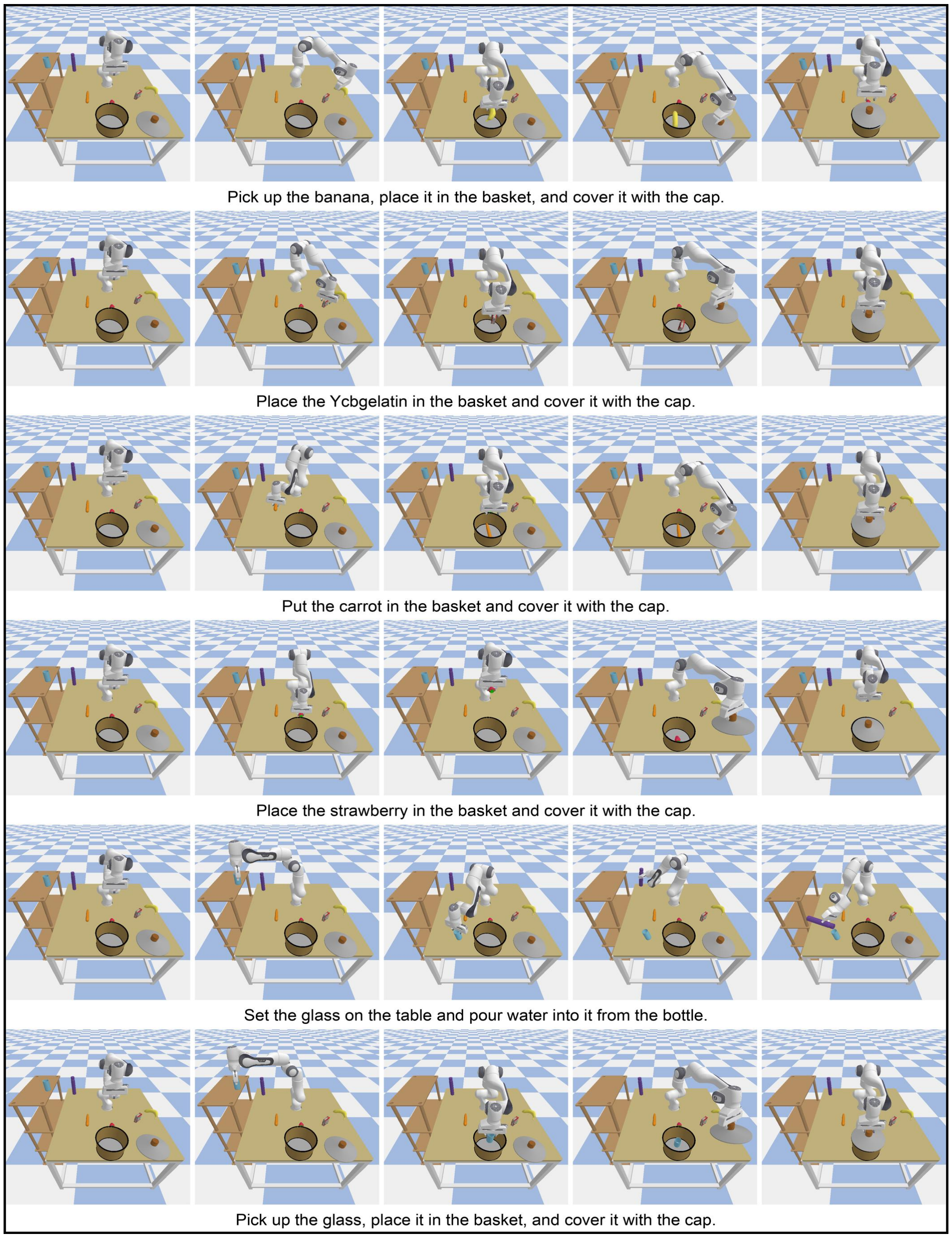
- *Domestic Assistive Tasks:* These simulate common household activities such as placing fruits in a basket or pouring water into a bottle, reflecting real-world assistive robotics applications.

Architecture	**Subgoal Transition Threshold**	**InD**		**OoD**	
		Success	**Completion**	**Success**	**Completion**
GPT	0.0008	**0.000**	0.111	**0.000**	0.106
GPT	0.0005	**0.000**	0.133	**0.000**	0.103
GPT	0.0001	**0.000**	0.175	**0.000**	0.067
GPT	0.00008	**0.000**	0.156	**0.000**	0.064
GPT	0.00005	**0.000**	0.131	**0.000**	0.039
ST-GPT (**Ours**)	N.A.	**0.000**	**0.242**	**0.000**	**0.181**

Table 6: Comparison between ST-GPT and GPT models with varying subgoal transition thresholds in the FrankaLHT environment. None of the models achieved a positive success rate, indicating that the very idea of discrete subgoal transitions is not strong enough for scaling to very long-horizon, complex tasks as posed by FrankaLHT. Nonetheless, the ST-GPT model demonstrates the highest completion rate in both in-distribution (Ind) and out-of-distribution (Ood) settings, highlighting its relatively better performance compared to the standard GPT model.

- *Medical Waste Sorting:* These tasks involve sorting and placing medical items—such as syringes and syrup bottles into appropriate trays and bins, relevant to hospital automation scenarios.
- *Food and Beverage Packaging:* These tasks are inspired by industrial packaging workflows, where the robot packs food items and beverage bottles into cardboard boxes or cartons.

Table 6 presents a comparison between ST-GPT and the standard GPT model in the FrankaLHT environment, similar to the analysis shown in Table 1 for the Franka Kitchen domain.



Pick up the banana, place it in the basket, and cover it with the cap.

Place the Ycbgelatin in the basket and cover it with the cap.

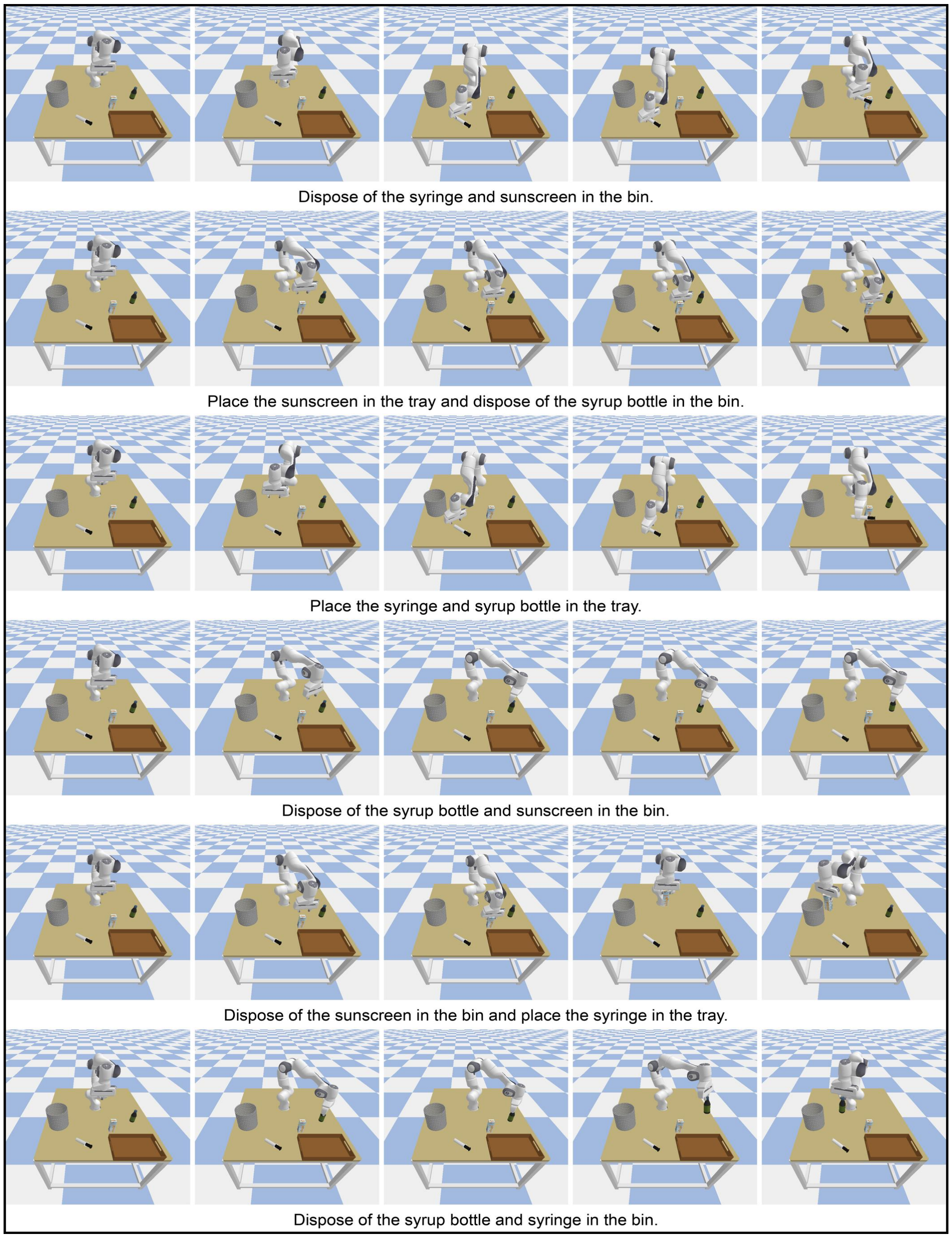
Put the carrot in the basket and cover it with the cap.

Place the strawberry in the basket and cover it with the cap.

Set the glass on the table and pour water into it from the bottle.

Pick up the glass, place it in the basket, and cover it with the cap.

Figure 8: Domestic Assistive Tasks (FrankaLHT)


Dispose of the syringe and sunscreen in the bin.

Place the sunscreen in the tray and dispose of the syrup bottle in the bin.

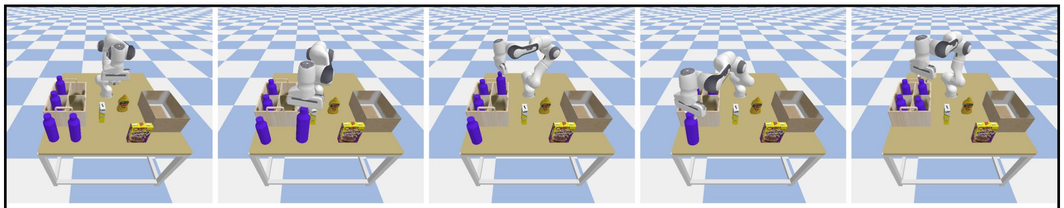
Place the syringe and syrup bottle in the tray.

Dispose of the syrup bottle and sunscreen in the bin.

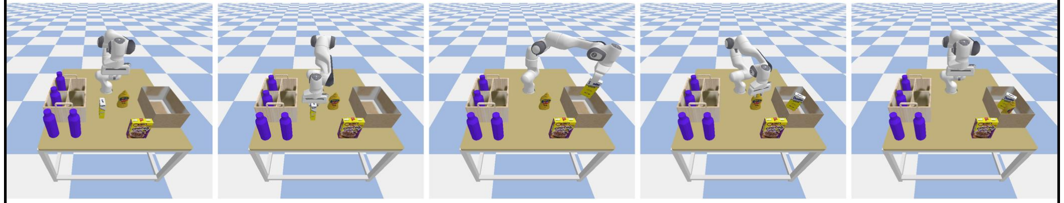
Dispose of the sunscreen in the bin and place the syringe in the tray.

Dispose of the syrup bottle and syringe in the bin.

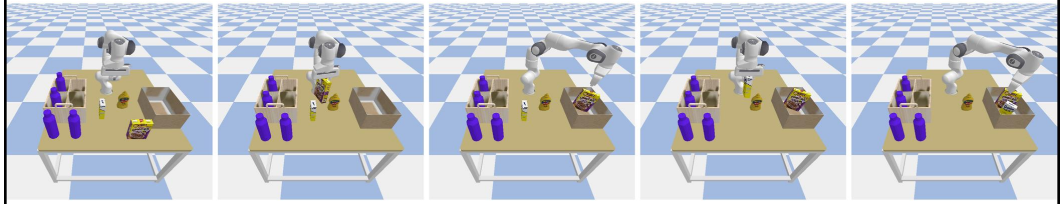
Figure 9: Medical Waste Sorting Tasks (FrankaLHT)



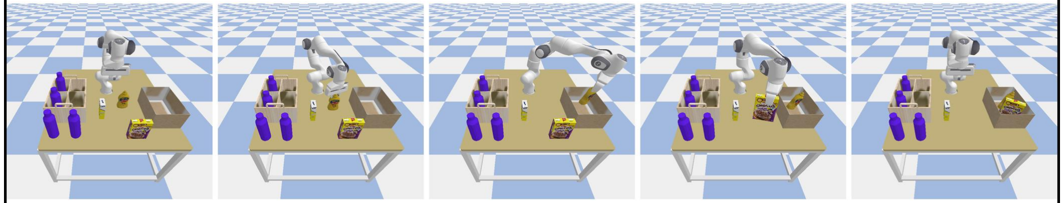
Pack the right bottle in the back slot and the left bottle in the front slot of the bottle packaging box.



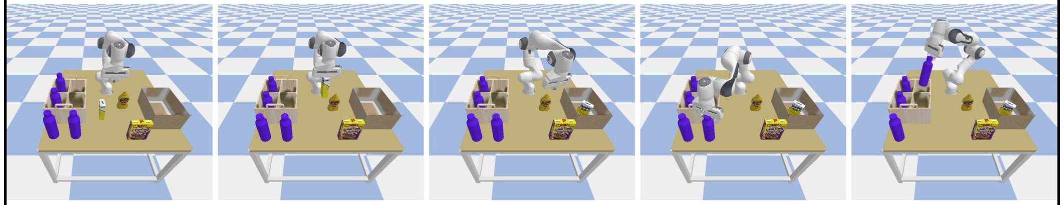
Pack the Domino Sugar and Mustard Bottle in the cardboard box.



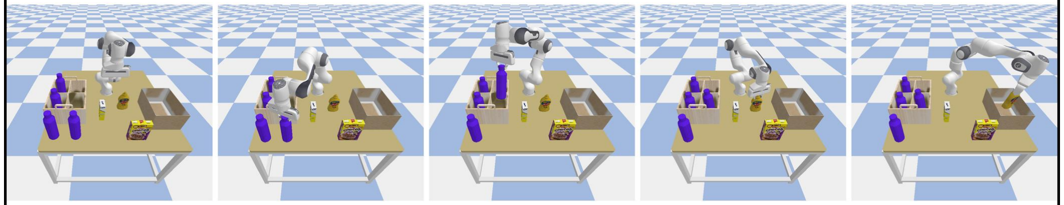
Pack the Kellogg's and Domino Sugar in the cardboard box.



Pack the Mustard Bottle and Kellogg's in the cardboard box.



Pack the Domino Sugar in the cardboard box and place the right bottle in the back slot of the bottle packaging box.



Pack the right bottle in the front slot of the bottle packaging box and the Mustard Bottle in the cardboard box.

Figure 10: Food and Beverage Packaging Tasks (FrankaLHT)

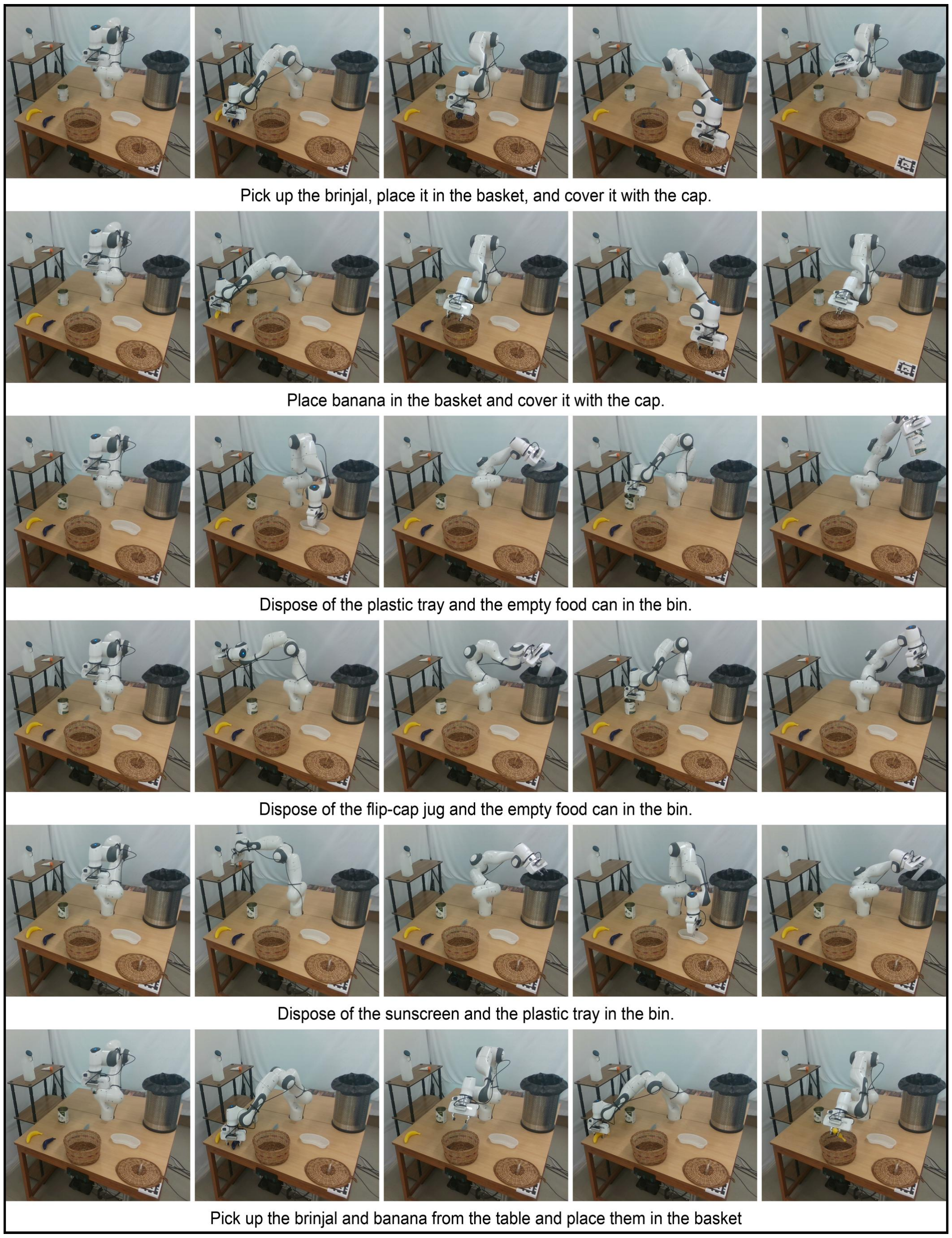

Pick up the brinjal, place it in the basket, and cover it with the cap.
Place banana in the basket and cover it with the cap.
Dispose of the plastic tray and the empty food can in the bin.
Dispose of the flip-cap jug and the empty food can in the bin.
Dispose of the sunscreen and the plastic tray in the bin.
Pick up the brinjal and banana from the table and place them in the basket

Figure 11: Real-world setup showing all 6 table-top manipulation tasks.