

LLM-Personalize: Aligning LLM Planners with Human Preferences via Reinforced Self-Training for Housekeeping Robots

Dongge Han¹, Trevor McInroe, Adam Jelley, Stefano V. Albrecht, Peter Bell, Amos Storkey

Abstract—Large language models (LLMs) have shown significant potential for robotics applications, particularly task planning, by harnessing their language comprehension and text generation capabilities. However, in applications such as household robotics, a critical gap remains in the personalization of these models to individual user preferences. We introduce LLM-Personalize, a novel framework with an optimization pipeline designed to personalize LLM planners for household robotics. Our LLM-Personalize framework features an LLM planner that performs iterative planning in multi-room, partially-observable household scenarios, making use of a scene graph constructed with local observations. The generated plan consists of a sequence of high-level actions which are subsequently executed by a controller. Central to our approach is the optimization pipeline, which combines imitation learning and iterative self-training to personalize the LLM planner. In particular, the imitation learning phase performs initial LLM alignment from demonstrations, and bootstraps the model to facilitate effective iterative self-training, which further explores and aligns the model to user preferences. We evaluate LLM-Personalize on Housekeep, a challenging simulated real-world 3D benchmark for household rearrangements, and show that LLM-Personalize achieves more than a 30 percent increase in success rate over existing LLM planners, showcasing significantly improved alignment with human preferences. Project page: <https://donggehan.github.io/projectllmpersonalize/>.

I. INTRODUCTION

The application of large language models (LLMs) to the robotics domain has demonstrated substantial potential, especially in the realm of task planning [1], [2], [3], [4], [5], [6], [7], [8], by leveraging their advanced language comprehension and text generation capabilities. An important challenge of using LLM-powered planners is the alignment of the LLM with the specific task context. While many studies have focused on grounding LLM planners to the physical contexts of the tasks to ensure executability of the generated plans and their relevance to the environment, our work aims to further extend this foundation to study personalization, an important aspect to household robotics which tailors the functionality of the household robot to the unique and subtle household needs and preferences.

Prior works on LLM grounding have proposed to align LLMs with the tasks' physical context through methods such as translating generated plans to executable actions [4], integrating contextual information such as affordance [2], [8], scene graph [3] or environmental feedback [3], [7]. Despite these advancements, there is noticeable gap in grounding LLM planners to personalized user preferences due to the inherent misalignment between the general-purpose knowledge

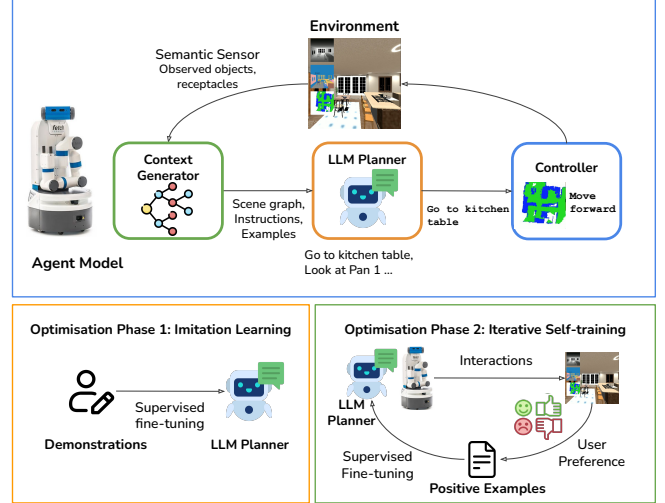


Fig. 1: Illustration of LLM-Personalize. Agent architecture: The Context Generator constructs and updates a scene graph from local observations. The LLM Planner uses the graph to produce a plan as a sequence of high-level actions, and iteratively re-plans when the previous plan has been executed. Each high-level action is translated to a sequence of control actions and executed by the Controller. To personalize the LLM Planner, we introduce an optimization pipeline integrating imitation learning and iterative self-training to fine-tune and align the planner with user preferences.

of LLMs, designed to reflect common preferences, and the unique preferences of users, e.g., one household may prefer a coffee mug to be placed on the dining table, whereas another may prefer for it to be in a kitchen cabinet.

To address this challenge, we propose LLM-Personalize, a household robotic agent framework that performs object rearrangements in multi-room and partially observable household scenarios. As shown in Fig. 1, the model integrates three key components: a context generator, an LLM planner, and a low-level controller. Central to personalizing the LLM planner to user preferences is our novel optimization pipeline that combines imitation learning with iterative self-training. In the first optimization phase, imitation learning is leveraged to bootstrap the model in order to 1) guide the LLM planner to interpret complex input contexts, 2) initial alignment of the planner's behavior with example user preferences, and 3) bootstrap the LLM planner to generate plans that can be clearly and straightforwardly annotated with user preferences, thus facilitating effective self-training in the second phase. Following this, an iterative self-training phase [9] is employed, enabling the LLM planner to further explore by

¹All authors are affiliated with the School of Informatics, University of Edinburgh, Edinburgh, UK. Correspond to dongge.han@ed.ac.uk.

collecting datasets of interactions, and optimize itself based on the positive interactions according to the user preferences.

We empirically evaluate our LLM-Personalize framework on the Housekeep benchmark [10], a challenging, long-horizon, partially observable household rearrangements task suite, featuring diverse house layouts and a wide variety of receptacles and objects. The quality of object rearrangements are assessed according to the rearrangement success according to the Housekeep benchmark based on their collected human preference data. We show in our experiments that LLM-Personalize outperforms state-of-the-art baseline LLM-based planners [1], [2], [3] with more than a 30 percent increase in success rate, as a result of improved understanding and alignment with human preferences. The inherent versatility and scalability of our framework render it suitable for diverse household layouts, object selections, and broader household robotics applications.

II. RELATED WORKS

A. LLM-Empowered Robotic Agents

The use of LLMs for building robotic agents has recently gained wide research interest. Prior works in task planning have effectively utilised pre-trained LLMs for generating executable plans for robotic agents [1], [2], [3], [4], [5], [6], [7], [8]. However, two key challenges that remain are the scalability of these methods to long-horizon planning tasks in large scenes, and misalignment of the LLM with the human preferences. [11] used LLMs for inferring rules summarizing personalized user preferences, in contrast, our work studies optimization and personalization of LLM planners for complex plan/action sequencing in multi-room household scenarios. Closely related to our work are [1], [2], [3]: [2] performs grounding of LLM planners with affordance functions. However, it is applicable to small scenes and limited vocabulary of objects. [3] addresses the scalability problem with a static 3D scene graph. [1] addresses the scalability problem by allowing LLMs to plan iteratively. In this work, we directly use LLM for plan generation similarly to [1]. To address the long-horizon planning and large scene problem, we allow the agent to start from an empty graph and dynamically update the graph as it explores the household, and iteratively re-plan when the current plan finishes. Moreover, we address the misalignment problem by developing an optimization pipeline which effectively aligns the LLM with personalized preferences.

B. Aligning LLMs with Human Preference

Alignment is the process of encoding human preferences into LLMs to ensure the LLMs generates contents consistent with human values [12]. Substantial progress have been made in LLM alignment including reinforcement learning (RL) based methods such as [13], [14], [9], [15], [16] and supervised learning (SL) based methods such as [17], [18], [19], [9], [20], and self-training [9] uses either an RL or SL objective. In this work, we adopt imitation learning (IL) to bootstrap our LLM planner to produce high-quality training examples for the self-training phase. Following

IL, we adapt self-training [9] to our LLM planner which iteratively explores and aligns itself with human preferences via supervised fine-tuning. In addition to the methodological considerations, our approach circumvents the limitations often encountered with using online blackbox LLMs, where RL based fine-tuning is not readily available.

III. METHOD

A. Preliminaries

1) *Large Language Model (LLM)*: An LLM [21] is a transformer-based [22] language model that is capable of performing general-purpose language generation, e.g., GPT [21], PaLM [23], LLaMA [24]. In practice, users interact with LLMs by sending *prompts* and receiving *responses*. Approaches to optimizing an LLM’s performance include prompting techniques such as in-context learning [21] or fine-tuning [25]. The linguistic abilities of LLMs enable a wide range of applications, e.g., in task planning, LLMs can leverage their capabilities to interpret complex context and generate detailed, coherent plans [1], [2], [3].

2) *Self-Training*: Iterative reinforced self-training [9] is a sample efficient algorithm for aligning LLMs with human preferences, particularly for the domain of machine translation. The key mechanism involves a *grow* step where a training dataset is collected by prompting the LLM to generate multiple responses for each prompt, and an *improve* step, where the dataset is annotated and filtered according to human preferences, followed by fine-tuning the LLM on the dataset using offline RL or supervised learning. For simplicity we will refer to the method as self-training (ST).

B. Problem Formulation

In this paper, we address the lack of personalization of LLM planners in household robotics tasks. To simulate a realistic household environment with user preferences, we employ Housekeep [10], a collection of 3D simulated household tasks, where a robotic agent is tasked to rearrange misplaced objects to suit collected user preferences. A *scene*, as shown in Fig. 2, refers to a household layout which includes a selection of *rooms* (e.g. a room might be a specific bedroom) and an arrangement of *receptacles* in each room. Let j count through each receptacle, and rec_j denote high level information about each receptacle, consisting of its unique identifier, its receptacle type, and the unique room and room type it is in. For example we might describe a rec_j as (*kitchen 0 table 6*) if it is the seventh receptacle in the household scene (counted from zero), is a table, and is in the first kitchen. Likewise, let i count through each of the objects, and let obj_i denote the high level information about an object consisting of its unique identifier, and its object type. For example, obj_i might be described as (*laptop 1*) if it is the first laptop in the scene.

Finally, we use M_j^t to denote the set of indices i of objects that are on a receptacle j at time t – object locations will change over time. This set can be empty, and for some receptacles, M_j^t may be constrained in cardinality. All the receptacles, objects and locations must be discovered by the

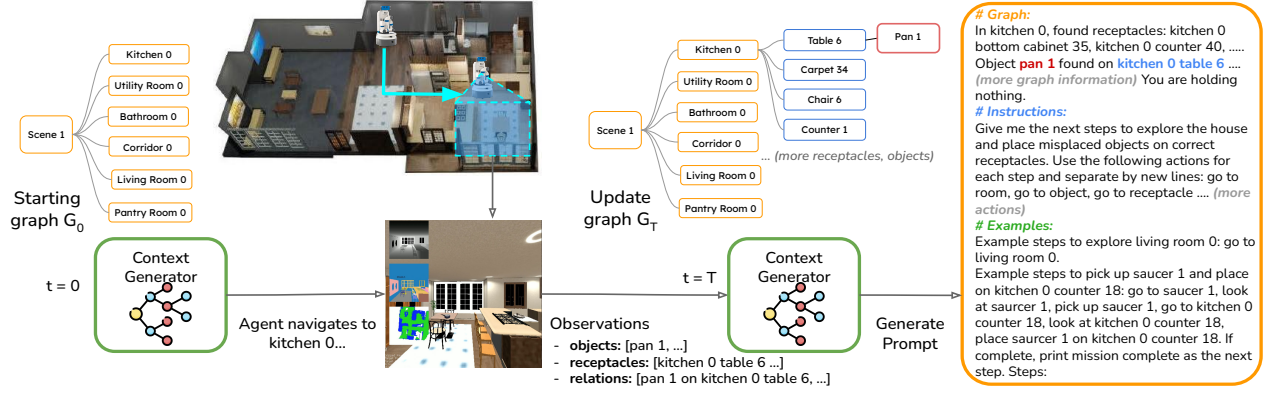


Fig. 2: The Context generator model maintains and updates the graph of the household state including rooms, receptacles and objects, derived from the robot’s local observations at each timestep. The information is provided as a prompt for the LLM planner. The top-down view of the scene is for illustration only, the robot only has access to the 1st-person view.

agent. Their existence is not given a priori. Each *task/episode* includes a scene with a random selection of objects, some of which are potentially misplaced in the wrong receptacles. The task the agent has to achieve is always communicated as “Give me the next steps to explore the house and place misplaced objects on correct receptacles”. The idea of misplacement has to be understood or learnt by the model.

At each timestep t , the robot receives an egocentric observation about a number of receptacles that are in the field of view, and any objects located therein. We collect the indices of the observed receptacles in R_t . So the observation at time t consists of the high level observation $o_t = \{(rec_j, obj_i \mid i \in M_j^t) \mid j \in R_t\}$, which is a list of the observed receptacles and the associated objects located on those receptacles, along with additional lower-level information, such as the locations of the objects etc. The robot, having the capacity of holding at most a single object, selects an action from its (low-level) action space $A = \{move\ forward, turn\ left, turn\ right, look\ up, look\ down, grab/release\}$. It receives a reward $+1$ for placing an object on a correct receptacle, -1 for grabbing/removing an already correctly placed object, else 0. The correctness of *obj-rec* placement is decided by a human preference dataset collected by Housekeep from human annotators. The robot’s learning goal is to find a control policy that maximizes cumulative rewards over the episode’s horizon of 1000 timesteps.

In this paper we will utilise a model that consists of a fixed low-level controller that handles the low level actions, and just work in a high-level space, for which an observation representation of the form of o_t is fully sufficient.

C. Model

Our robotic agent model is designed to perform long-horizon planning in the partially observable household scenarios by integrating three key components: the context generator, the LLM planner and the controller. Specifically, the context generator provides context information for decision-making in the form of prompts, by maintaining an internal representation of the household state derived from observations. For decision making, we choose a two-level design

that integrates a high-level planner powered by an LLM for generating high-level plans, and a controller which executes the generated plans using low-level control actions. Given our primary focus on personalizing LLM planners, we make use of an off-the-shelf controller from the simulator, and focus on designing the context generator and LLM planner.

1) *Context Generator*: The context generator provides information of the current household context and useful instructions as a prompt for the downstream LLM planner. Specifically, our context generator provides three pieces of information: the current household state, instructions, and examples for in-context learning [21]. As the agent never observes the full state of the environment, but only receives a partial view o_t , it is important that the agent maintains and refines an internal representation of the household state to correctly choose the placement of objects (e.g., only using local observations can lead to a suboptimal placement when the correct receptacle of a misplaced object is in a different room). To this end, our context generator maintains a graph G as shown in Fig. 2. When a task starts, the context generator initializes the graph G_0 of the house with empty room nodes. At each timestep t , the graph is updated with the locally observed objects and receptacles o_t as the agent navigates around the house. To provide the information for decision-making, a prompt is constructed with a natural language description of G_t , the object held by the robot, instructions (a description of the overall rearrangement task, the role assigned to the LLM, and the available high-level actions), and two examples, one for room exploration and another one for moving an object from one receptacle to another, allowing the LLM planner to follow via in-context learning.

2) *LLM Planner*: The LLM planner is the core decision-making module that generates a *high-level plan as a sequence of high-level actions*, and each high-level action is translated by the low-level controller to low-level actions and executed. The set of available *high-level actions* are: $\Omega = \{go\ to\ obj/rec/room, look\ at\ obj/rec, pick\ up\ obj, place\ obj\ on\ rec\}$, where the *obj*, *rec*, *room* are replaced by the actual names of these target entities, as introduced in section III-B.

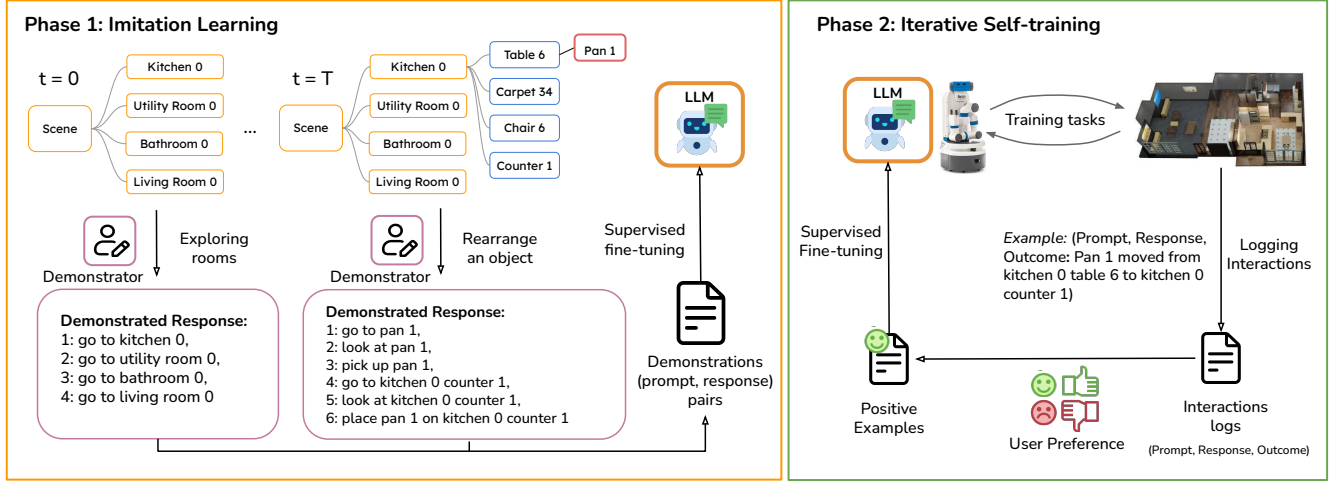


Fig. 3: Optimization pipeline of LLM-Personalize using imitation learning and iterative self-training.

To handle partial observability in the multi-room households, we adopt an iterative planning procedure which enables the LLM planner to re-plan when the previous plan has finished execution. This allows the agent to obtain a more comprehensive understanding of the household state while exploring and navigating the house, leading to improved rearrangement decisions. On the other hand, compared with single-step iterative planning where each plan includes only one high-level action, our approach builds cohesive plans that better account for the inter-dependencies and the cumulative effect of the action sequences. The iterative procedure works as follows: Denote $n \in \mathbb{N}$ as a high-level planning iteration. At iteration n , the LLM planner receives a prompt from the context generator and generates an immediate action *plan* as a sequence of high-level actions $\omega \in \Omega$: $p_n = (\omega_0, \omega_1, \dots)$ (see Fig. 3 for two example plans). The plan is sent to the controller where the high-level actions are executed sequentially. Each high-level action is translated to a sequence of low-level control actions $a \in A$, e.g., *go to pan 1* $\rightarrow$ (*move forward, turn left, ...*). Once the controller finished executing all high-level actions in p_n , say at timestep $t = T$, the next plan iteration $n + 1$ starts where the LLM planner is prompted again to generate a new plan p_{n+1} and executed by the controller. This process is repeated iteratively.

The LLM planner is implemented with an LLM model and a post-processor. Upon receiving a prompt from the context generator, the LLM returns a plan in natural language as a sequence of high-level actions: *go to pan 1, pick up pan 1, ...*. Specifically, the first 10 high-level actions from the sequence are used, as we often observe a decrease in quality towards the end of a long response from the LLM. The post-processor then extracts from each high-level action the target action (i.e., one of *go to, look at, pick up, place*) and the target entities (i.e., *obj, rec, room*) and send to the controller to be translated and executed as low-level control actions.

3) *Controller*: Given our primary focus on personalizing LLM planners, we assume a room-level topological map is available (e.g., via semantic mapping [26]). We make use of the off-the-shelf controller accessible from the Housekeep

simulator. During a task, the robot navigates and continually updates the map using egocentric observations, camera projection matrix, RGBD-aligned pixel-wise instance and semantic masks and relationship sensor to localize objects and receptacles to update the map. The high-level actions from the LLM planner are carried out as follows: 1) for *go to*, the controller uses the allocentric map and the target entity (*obj, rec, room*), and executes a sequence of navigation actions to reach the target; 2) for *look at*, the agent orients itself to face the desired target via look up/down and turn left/right actions; 3) to carry out *pick/place*, the agent invokes a discrete grab/release action that casts a ray, and if it intersects an *obj* or *rec* within 1.5m, it picks or places an object. More details of the controller can be found in [10].

D. Personalizing the LLM Planner

Despite our model architecture being well-suited to the partially observable household scenarios, we observed two challenges that necessitated a tailored optimization process. 1) LLM planners struggle with effectively extracting precise information from complex input contexts (e.g., resulting in plans with partial object names). This is compounded by the complexity of accurately sequencing high-level actions to ensure executability. 2) Misalignment between the LLM planners' decisions and the personalized preferences of users.

Self-training provides a promising approach to optimizing and personalizing the LLM planner with user preferences. However, direct application of self-training methodologies to the LLM planner presents new, unique challenges: Unlike single-step generation tasks like machine translation, the household robotics tasks often involve long-horizon planning, where the LLM planner may generate a plan consisting of both correct and incorrect placements actions, making it difficult to annotate with human preferences and extract clean training examples for automatic self-training.

To this end, we introduce a tailored optimization pipeline that integrates imitation learning and iterative self-training. The imitation learning phase bootstraps the LLM planner to effectively interpret the complex context, produce executable

plans, and perform initial alignment with example user preferences. Moreover, the demonstrations are designed to bootstrap the LLM planner to generate plans that can be clearly annotated, thus facilitating effective self-training in the second phase. Following this, the iterative self-training phase allows the LLM planner to further explore and refine its planning strategies based on user preferences.

1) *Imitation Learning*: As shown in Fig. 3, we build a demonstrator module to generate demonstrated responses for the LLM planner on a set of demonstration tasks. Upon receiving a prompt $\mathbf{x}$ from the context generator, the demonstrator produces a plan $\mathbf{y}$ which performs either exploration of the rooms or rearrangement of a single object, using the scene graph from the context generator and the correct object-receptacle mapping according to the human preference dataset (described in section III-B). Specifically, when prompted at the start of a task, the demonstrator produces a plan of high-level actions to visit each of the rooms. After the plan is executed, the agent will have discovered some misplaced objects and receptacles in each room. The demonstrator will be prompted again to generate a plan, which rearranges one of the discovered misplaced object (picked randomly) to a discovered correct receptacle. The plan is executed by the controller, and we iterate the procedure until all discovered objects are correctly placed.

To bootstrap the LLM planner, we prepare the collected demonstrations as pairs of prompts and target responses, $\mathcal{D}_{\text{demo}} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$, where both $\mathbf{x}^i, \mathbf{y}^i$ are sequences of tokens, the response $\mathbf{y}^i$ being a plan by the demonstrator. Given a pre-trained autoregressive LLM $P_\theta(\mathbf{y} | \mathbf{x})$ parametrized by θ , we perform supervised fine-tuning of the LLM model on $\mathcal{D}_{\text{demo}}$ by minimizing the negative log likelihood (NLL) loss:

$$\mathcal{L}_{\text{NLL}} = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{demo}}} \left[\sum_{\tau=1}^{|\mathbf{y}|} \log P_\theta(\mathbf{y}_\tau | \mathbf{y}_{1:\tau-1}, \mathbf{x}) \right] \quad (1)$$

The above demonstration design guides the LLM planner to more accurately extract information from complex input contexts and improve plan executability. More importantly, it ensures that each plan has a uniform objective (i.e., perform exploration or rearrangement of a single object), allowing the plan to be straightforwardly annotated with user preferences.

2) *Iterative Self-training*: Next, we fine-tune the bootstrapped LLM planner to further improve personalization via iterative self-training on a set of training tasks. This allows the LLM planner to explore more rearrangement options and improve its placement decisions through imitating the positive examples. As shown in Fig. 3, to start a self-training iteration, we use the LLM planner to explore (with randomness by adjusting the temperature parameter) by collecting episodes of experiences on the training tasks and we log M interactions as tuples of prompt, response and outcomes: $\{(\mathbf{x}^i, \mathbf{y}^i, \text{out}^i)\}_{i=1}^M$ (Example outcome: pan 1 moved from kitchen 0 table 6 to kitchen 0 counter 1). We annotate each prompt and response pair $(\mathbf{x}^i, \mathbf{y}^i)$ with the reward $r^i \in \{1, 0, -1\}$ for the rearrangement outcomes according to the user preferences. Then, we collect a self-training dataset by picking the prompt and response pairs

with positive rewards $\mathcal{D}_{\text{self-train}} = \{(\mathbf{x}^i, \mathbf{y}^i) | r^i > 0\}_{i=1}^M$. The final step of this self-training iteration is to perform supervised fine-tuning of the bootstrapped LLM model over $\mathcal{D}_{\text{self-train}}$ with the NLL objective as defined in Equation (1). This procedure is repeated iteratively where each self-training iteration performs interaction collection and fine-tuning over the LLM obtained from the previous iteration.

IV. EXPERIMENTS

A. Overview

We empirically evaluate the performance of LLM-Personalize on the Housekeep benchmark. In particular, we aim to evaluate the hypothesis: *Optimizing LLM planner through imitation learning and self-training allows the LLM planner to improve planning performance and alignment with user preferences*. We demonstrate this hypothesis through improved rearrangement success rate compared with baseline LLM planners and provide qualitative results that showcase the plans generated. Our ablations studies further evaluate plan executability, exploration and cross-domain (scene) generalisation of LLM-Personalize in the different training phases. These analyses provide a deeper dive into the improved task success and offer more insights for personalizing LLM planner in future applications.

B. Experiment Setup

For our main results, we evaluate LLM-Personalize on in-domain (scene) adaptation performance on 4 different scenes in the Housekeep benchmark.

1) *Environment setup*: Each scene contains a different layout of rooms and receptacles, as shown in Fig. 4. A *task* is instantiated with a random selection of 5 – 10 objects placed on different receptacles, among them $\sim 3 - 7$ objects are misplaced and needs to be rearranged. For each object, there is a list of correct receptacles not known to the agent. The task is challenging since the agent needs to explore the house, identify misplaced objects and their correct receptacles, and avoid moving correctly placed objects to wrong receptacles.

2) *Evaluation metrics*: We evaluate the improvement of success rate of in-domain adaptation. The success rate [10] is defined as the percentage of misplaced objects that are correctly re-arranged at the end of the task, among all misplaced objects at the start of the task.

$$\text{Success Rate} = \left(\frac{\# \text{correct at the end} - \# \text{correct at the start}}{\# \text{total misplaced objects at the start}} \right)$$

With the success rate defined in terms of difference, an agent is judged fairly for its correct and wrong placements: an agent which performed poorly that resulted in more misplaced objects at the end of the task will yield a negative success rate. We measure the success rate on train and test tasks at each phase of the optimization pipeline. For each scene, we randomly sample three disjoint set of 10 demonstration tasks, 20 training tasks and 5 test tasks. Hence the agent will encounter a random selection of objects and placement configurations on each task. During both training and testing, we collect experiences of 5 episodes per task,

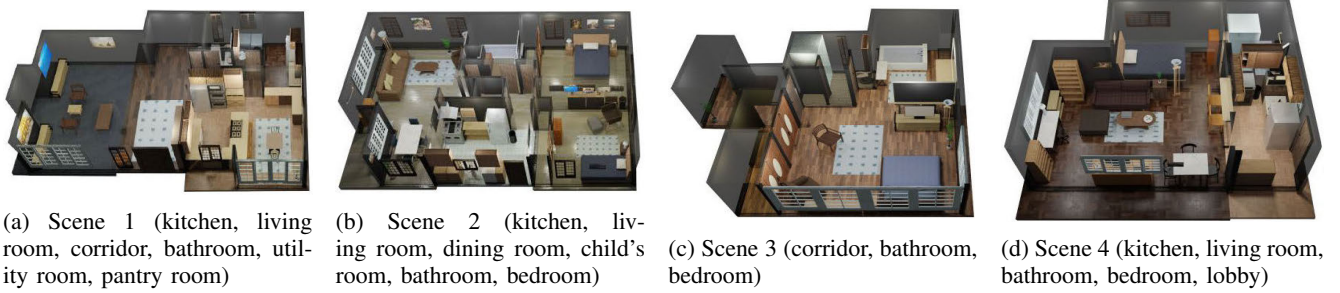


Fig. 4: The Housekeep scenes used in our experiment.

TABLE I: Average success rate on train and test sets across scenes. Each entry denotes the mean $\pm$ standard error of the mean across episodes. (**Boldface**: best variant across the task set, **ST**: self-training, **IL**: imitation learning)

	Scene ID	Scene 1		Scene 2		Scene 3		Scene 4		Average
	Task Set	<i>train</i>	<i>test</i>	<i>train</i>	<i>test</i>	<i>train</i>	<i>test</i>	<i>train</i>	<i>test</i>	<i>test</i>
Baselines	SayCan	-2.6 $\pm$ 1.9	0.0 $\pm$ 0.0	-1.2 $\pm$ 1.2	-10.6 $\pm$ 6.8	-3.3 $\pm$ 2.2	-8.0 $\pm$ 4.9	-1.6 $\pm$ 1.6	0.0 $\pm$ 0.0	-4.6 $\pm$ 2.9
	SayPlan	-7.0 $\pm$ 5.9	-5.0 $\pm$ 5.0	-6.8 $\pm$ 2.7	-1.6 $\pm$ 9.2	0.4 $\pm$ 4.5	-13.0 $\pm$ 8.3	-10.7 $\pm$ 3.8	-12.3 $\pm$ 15.3	-7.9 $\pm$ 9.4
	LLM-Planner	5.3 $\pm$ 4.4	-3.6 $\pm$ 4.8	-8.4 $\pm$ 3.8	-9.8 $\pm$ 6.3	-14.2 $\pm$ 4.0	-4.0 $\pm$ 3.3	-29.6 $\pm$ 5.2	-30.2 $\pm$ 4.8	-11.8 $\pm$ 4.8
Ours	LLM-Personalize (IL)	4.1 $\pm$ 2.6	17.6 $\pm$ 6.1	-3.3 $\pm$ 3.0	12.6 $\pm$ 9.0	22.6 $\pm$ 2.6	24.3 $\pm$ 4.4	10.9 $\pm$ 3.3	25.7 $\pm$ 6.7	20.0 $\pm$ 6.5
	LLM-Personalize (ST iter=1)	17.9 $\pm$ 3.7	25.8 $\pm$ 6.6	19.4 $\pm$ 2.9	21.7 $\pm$ 5.6	32.4 $\pm$ 3.2	41.4 $\pm$ 6.3	24.2 $\pm$ 3.2	10.2 $\pm$ 6.7	24.7 $\pm$ 6.3
	LLM-Personalize (ST iter=2)	25.5 $\pm$ 3.1	29.6 $\pm$ 5.4	18.5 $\pm$ 2.6	25.2 $\pm$ 4.0	33.5 $\pm$ 3.8	43.3 $\pm$ 4.3	29.1 $\pm$ 2.9	20.4 $\pm$ 6.8	29.6 $\pm$ 5.1

and present the mean and standard error of the metrics across all collected episodes in the task sets.

3) *Architecture and Baselines*: We compare LLM-Personalize with state-of-the-art LLM planning baseline methods: LLM-Planner [1], SayPlan [3] and SayCan [2]. We use GPT-3.5-turbo as the LLM across all compared methods, and the temperature during LLM generation is set to 1 to reduce deterministic repetition in LLM responses (default range is 0 – 2). For all methods, the prompt includes the instruction, graph description and two examples. We allow all methods to plan iteratively after the robot executed the previous plan. LLM-Planner with these configurations is adopted as the base model of LLM-Personalize, which we then optimize using imitation learning and self-training. The fine-tuning as defined in Eq.(1) is performed via the OpenAI fine-tune API. To adapt SayCan to Housekeep with large number of available actions, we adopt the implementation in [1], where the list of affordable high-level actions (e.g., go to a discovered object) are provided in each prompt, together with the list of previously executed actions. For SayPlan, we additionally provide the state and affordance of the receptacles and objects (e.g., pick up) as in [3] and allow 10 LLM semantic search steps and 5 revision (re-plan) retries for each plan iteration where the final revised plan is executed, and the revision feedback is provided by a verification module (e.g., cannot pick up *obj* as the agent is holding another object).

C. Main Results

1) *Quantitative Results*: We compare the performance in terms of average success rate of LLM-Personalize with the baseline methods in Table I, and we also demonstrate the

effectiveness of our optimization framework by comparing the performance of LLM-Personalize at different optimization phases, namely, the base version (using LLM-Planner), imitation learning (IL), and various iterations of self-training (ST). Each table entry shows the mean and standard error of the mean across all tasks in the train/test set and 5 runs per task. Overall, we observe that LLM-Personalize significantly outperforms all baseline methods in terms of mean success rate across the tested scenes. For example, on the test set of Scene 1, the success rate of the baselines are near zero or negative, while LLM-Personalize achieved 29.6% after imitation learning and two self-training iterations. This trend can be similarly observed across all scenes. To examine closely into the difference, we analysed some detailed prompt and responses and identified that SayCan often has difficulty picking the best (high-level) action from a large number of available actions due to the large number of objects, receptacles and rooms. LLM-Planner is often able to produce correct pick and place action sequences following the in-context examples. Compared with LLM-Planner, SayPlan improves slightly (on 5 out of 8 task sets) as a result of improvement in plan executability due to revision with feedback. However, all baselines have difficulty knowing whether an object is misplaced or correctly placed, as well as the correct receptacles to place objects, due to lack of personalization. As a result, the negative scores across the baseline methods are often due to picking and placing correctly placed objects onto wrong receptacles.

Comparing the different stages of LLM-Personalize on the test sets, we observe a general trend where the combination of imitation learning and self-training lead to better

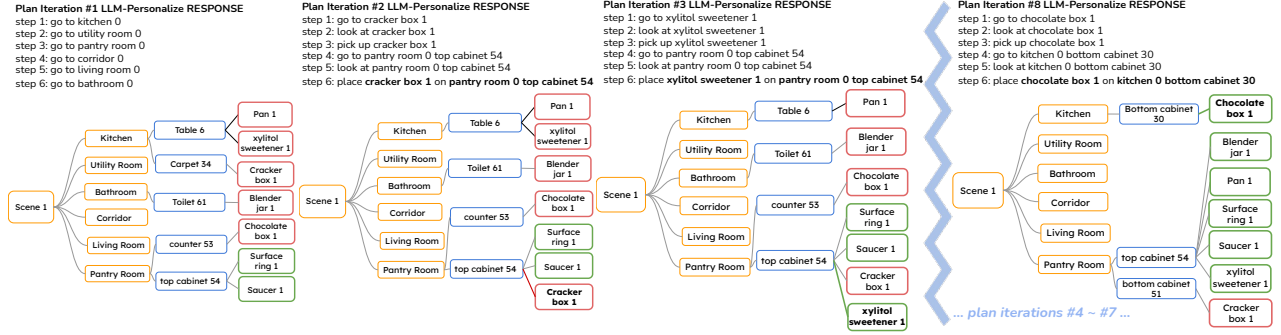


Fig. 5: Demonstration of four planning iterations generated and executed by LLM-Personalize (top row) and the resulting graphs (bottom row) on a test task in Housekeep. Green/red object (leaf) nodes indicate correct/wrong placements. The object being moved is shown in boldface with highlighted edge. This episode starts with 2 correctly placed objects and 5 misplaced objects (left), and changed to 6 correctly placed objects and only 1 misplaced objects after rearrangements (right). For clarity, the graphs only show receptacles with objects and omit all other receptacles.

TABLE II: Ablation Study: Cross-domain generalisation success rate. (**Boldface**: best across the task set, **ST1/2**: self-training iteration 1 or 2, **IL**: imitation learning)

Scene Pairs	Scene 1 & 2		Scene 3 & 4	
Task Set	<i>train(Scene2)</i>	<i>test(Scene1)</i>	<i>train(Scene4)</i>	<i>test(Scene3)</i>
LLM-Planner	-8.4 ± 3.8	-3.6 ± 4.8	-29.6 ± 5.2	-4.0 ± 3.3
LLM-Personalize(IL)	-3.3 ± 3.0	13.2 ± 6.5	10.9 ± 3.3	34.3 ± 5.4
LLM-Personalize(ST1)	19.4 ± 2.9	13.0 ± 4.6	24.2 ± 3.2	31.2 ± 4.9
LLM-Personalize(ST2)	18.5 ± 2.6	17.0 ± 4.5	29.1 ± 2.9	42.4 ± 3.8

results. First, bootstrapping from demonstrations improves over LLM-Planner (i.e., the base LLM-Personalize model). For example, on Scene 1, the success rate improved from -3.6% to 17.6%. This improvement is a result of improved executability due to action sequencing, better context understanding (e.g., agent correctly extracts and uses object names from the prompt), and initial alignment to personalized preferences shown in the demonstrations. Second, the self-training iterations further improves from the bootstrap variants with improved alignment with personalized preferences. For example, on Scene 1, the success rate after two iteration of self-training grows from 17.6% to 29.6% compared with the imitation bootstrapped variant. We also observed that after bootstrapping, the model learns to explore object placements with improved accuracy during self-training and with more self-training iterations, the model exploits and commits to the learned correct placements while avoiding wrong ones.

Comparing the train and test performances of LLM-Personalize for each scene, we observe that the testing performance generally increases with increased training performance, except for Scene 3, where the test performance drops as a result of overfitting. This shows that our model is able to learn personalized preferences seen during training, and generalize to unseen object combinations and placements.

2) *Qualitative Results*: In Fig. 5, we present plans generated by LLM-Personalize and executed by the robot on a test task in the Housekeep simulator and the resulting graph of the household scene after each plan iteration. We can

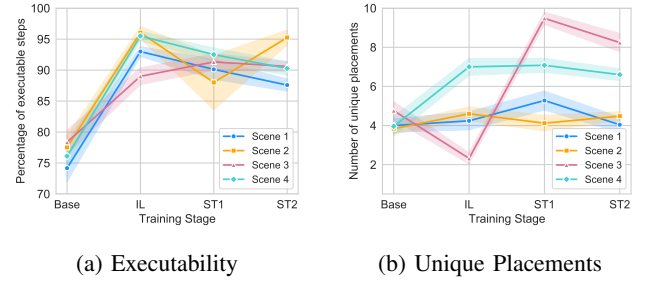


Fig. 6: Ablation study: (a) shows the percentage of executable high-level action steps, (b) shows the unique placements executed. x-axis refers to LLM-Personalize at different optimization phases – Base: before optimization, IL: imitation learning, ST: self-training. Each point is an average value over 25 episodes (5 runs per task over 5 tasks in the test set) and shaded area refers to standard error of the mean.

see that the agent learned to start by exploring the house, then rearrange one misplaced object at a plan iteration, and successfully rearranged 4 out of 5 misplaced objects.

D. Ablation Studies

In this section, we present ablation studies on LLM-Personalize’s plan executability, exploration vs. exploitation and cross-domain transfer performance.

1) *Cross-domain Transfer Results*: In addition to the improved in-domain adaptation results in Table I, we show in Table II cross-domain transfer performance of LLM-Personalize. In this experiment, we train the model on a source scene (e.g., scene 2 or scene 4), and observe the performance change on the test set on a different scene (e.g., scene 1 or scene 3) with different rooms and receptacles. From the table, we can observe that through imitation learning and self-training, LLM-Personalize is able to transfer to a different scene with improved test performance.

2) *Executability*: In Fig. 6a we present the executability improvement of LLM-Personalize (IL, ST1, ST2) compared to the base LLM-Planner (Base). Each point refers to the average percentage of high-level actions generated by the LLM planner that are successfully executed per episode. As ex-

pected the IL bootstrapping significantly improved the planner’s executability, due to improved context (prompt) understanding and action sequencing, enabling LLM-Personalize to produce high quality training examples for self-training, and we can observe consistently high executability from the LLM-Personalize(ST1) and LLM-Personalize(ST2) variants.

3) *Exploration vs. Exploitation*: To analyse how the agent’s exploration vs. exploitation behavior changes at different optimization phases, we show in Fig. 6b the average number of unique placements executed per episode, where each unique placement refers to a pair of object and receptacle where the agent placed the object on the receptacle. Higher degree of exploration behavior is indicated by higher number of unique placements, and in contrary, lower number of unique placements indicates more exploitation behavior. As shown in Fig. 6b, we see an increasing trend in exploration from LLM-Planner (Base) to LLM-Personalize(IL), partly due to improved plan executability. From IL to ST1, the agent further increased exploration, where it explores placements beyond the behaviors learned from demonstrations. For example, on Scene 3 the average unique placements increased from 2.32 to 9.48. Compared with ST1, ST2 typically shows higher exploitation behavior where the agent learns to commit to the correct placement combinations for better task performance.

V. CONCLUSIONS

We proposed LLM-Personalize, a household robotics agent framework with an LLM-based architecture capable of performing long-horizon planning in multi-room, partially observable household scenarios, and an optimisation pipeline designed to personalize the LLM planner according to user preferences. This novel approach effectively addresses the critical gap in the personalization of LLM planners for household robotics. Our model achieves superior alignment with user preferences, outperforming existing work in the challenging Housekeep rearrangement tasks. Looking forward, the versatility and scalability of the agent design and optimization pipeline makes LLM-Personalize a promising solution for a broad range of household robotics applications.

REFERENCES

- [1] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, “Llm-planner: Few-shot grounded planning for embodied agents with large language models,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2998–3009.
- [2] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [3] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Sunderhauf, “Sayplan: Grounding large language models using 3d scene graphs for scalable task planning,” *arXiv preprint arXiv:2307.06135*, 2023.
- [4] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.
- [5] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.
- [6] J. Mai, J. Chen, B. Li, G. Qian, M. Elhoseiny, and B. Ghanem, “Llm as a robotic brain: Unifying egocentric memory and control,” *arXiv preprint arXiv:2304.09349*, 2023.
- [7] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [8] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman, *et al.*, “Grounded decoding: Guiding text generation with grounded models for robot control,” *arXiv preprint arXiv:2303.00855*, 2023.
- [9] C. Gulcehre, T. L. Paine, S. Srinivasan, K. Konyushkova, L. Weerts, A. Sharma, A. Siddhant, A. Ahern, M. Wang, C. Gu, *et al.*, “Reinforced self-training (rest) for language modeling,” *arXiv preprint arXiv:2308.08998*, 2023.
- [10] Y. Kant, A. Ramachandran, S. Yenamandra, I. Gilitschenski, D. Batra, A. Szot, and H. Agrawal, “Housekeep: Tidying virtual households using commonsense reasoning,” in *European Conference on Computer Vision*. Springer, 2022, pp. 355–373.
- [11] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, “Tidybot: Personalized robot assistance with large language models,” *arXiv preprint arXiv:2305.05658*, 2023.
- [12] T. Shen, R. Jin, Y. Huang, C. Liu, W. Dong, Z. Guo, X. Wu, Y. Liu, and D. Xiong, “Large language model alignment: A survey,” *arXiv preprint arXiv:2309.15025*, 2023.
- [13] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, “Training language models to follow instructions with human feedback,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.
- [14] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *arXiv preprint arXiv:2305.18290*, 2023.
- [15] A. Glaese, N. McAleese, M. Trebacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick, P. Thacker, *et al.*, “Improving alignment of dialogue agents via targeted human judgements,” *arXiv preprint arXiv:2209.14375*, 2022.
- [16] A. F. Akyurek, E. Akyurek, A. Madaan, A. Kalyan, P. Clark, D. Wijaya, and N. Tandon, “RL4f: Generating natural language feedback with reinforcement learning for repairing model outputs,” *arXiv preprint arXiv:2305.08844*, 2023.
- [17] H. Dong, W. Xiong, D. Goyal, R. Pan, S. Diao, J. Zhang, K. Shum, and T. Zhang, “Raft: Reward ranked finetuning for generative foundation model alignment,” *arXiv preprint arXiv:2304.06767*, 2023.
- [18] C. Xu, Z. He, Z. He, and J. McAuley, “Leashing the inner demons: Self-detoxification for language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 11 530–11 537.
- [19] H. Liu, C. Sferrazza, and P. Abbeel, “Chain of hindsight aligns language models with feedback,” *arXiv preprint arXiv:2302.02676*, vol. 3, 2023.
- [20] J. Scheurer, J. A. Campos, T. Korbak, J. S. Chan, A. Chen, K. Cho, and E. Perez, “Training language models with language feedback at scale,” *arXiv preprint arXiv:2303.16755*, 2023.
- [21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [23] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, “Palm: Scaling language modeling with pathways,” *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.
- [24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [25] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.
- [26] A. Pronobis and P. Jensfelt, “Hierarchical multi-modal place categorization,” in *ECMR*, 2011, pp. 159–164.