MOSAIC: A Modular System for Assistive and Interactive Cooking

Huaxiaoyue Wang*, Kushal Kedia*, Juntao Ren*, Rahma Abdullah, Atiksh Bhardwaj, Angela Chao, Kelly Y Chen, Nathaniel Chin, Prithwish Dan, Xinyi Fan, Gonzalo Gonzalez-Pumariega, Aditya Kompella, Maximus Adrian Pace, Yash Sharma, Xiangwan Sun, Neha Sunkara, Sanjiban Choudhury

Cornell University

https://portal-cornell.github.io/MOSAIC/

Fig. 1: **MOSAIC cooking in the kitchen.** (top) MOSAIC interacts with a user via natural language and controls a tabletop manipulator (R1) and a mobile manipulator (R2) to prepare vegetable soup with the user. (bottom) We evaluate MOSAIC on multiple recipes, involving a range of robot skills that interact with the human user and everyday objects.

***Abstract*—We present MOSAIC, a modular architecture for home robots to perform complex collaborative tasks, such as cooking with everyday users. MOSAIC tightly collaborates with humans, interacts with users using natural language, coordinates multiple robots, and manages an open vocabulary of everyday objects. At its core, MOSAIC employs modularity: it leverages multiple large-scale pre-trained models for general tasks like language and image recognition, while using streamlined modules designed for task-specific control. We extensively evaluate MOSAIC on 60 end-to-end trials where two robots collaborate with a human user to cook a combination of 6 recipes. We also extensively test individual modules with 180 episodes of visuomotor picking, 60 episodes of human motion forecasting, and 46 online user evaluations of the task planner. We show that MOSAIC is able to efficiently collaborate with humans**

* Denotes equal contribution.
Correspondence to: yukiwang@cs.cornell.edu

by running the overall system end-to-end with a real human user, completing $68.3\%(41/60)$ collaborative cooking trials of 6 different recipes with a subtask completion rate of 91.6%. Finally, we discuss the limitations of the current system and exciting open challenges in this domain. The project's website is at https://portal-cornell.github.io/MOSAIC/

I. Introduction

Collaborative tasks in home environments requiring a coordinated medley of skills pose significant challenges for robots. These tasks require robots to have natural interactions with human users, possess the ability to learn a diverse set of skills, and perform them in a collaborative manner. Prior systems in this domain [1–4] have demonstrated impressive capabilities. However, they typically have one of two limitations: either they operate in isolation and lack meaningful collaboration with humans, or they interact with humans in a highly scripted manner, and are therefore only capable of completing a narrow set of predefined tasks. In this paper, we aim to overcome both of these limitations by designing a system that fluidly collaborates with humans and performs a wide range of tasks.

We identify three key desiderata for the system: *(1)* interact with users via natural language, *(2)* perform a range of skills that require manipulating everyday objects, and *(3)* collaborate seamlessly with humans. Consider the scenario in Figure 1, where a human user collaborates with two robots to prepare a meal. The user should be able to effortlessly interact with the system via natural language to decide on a new recipe. The robots in turn should perform the necessary skills to make the recipe, such as fetching a range of ingredients and cooking with them. Finally, the robots must fluidly collaborate with humans, such as handing over items.

One of the key challenges in building a collaborative agent that functions seamlessly in the wild is ensuring that it is able to act safely across an expansive set of possible inputs. While a single end-to-end model works well for tasks like language understanding where large amounts of data are available, such an approach is difficult for robot controls, where less data is available and extreme precision is important. Our key insight is that ***by modularizing our architecture, we can segment out parts of the framework that require broad generalization, such as language and image recognition, from the portions that require task-specific control.*** This division of work means that strong overall performance can be achieved through specialization: we can use large *pre-trained models* to extract useful information from large and unstructured input spaces and *task-specific models* to make safe and precise decisions.

We apply this modular approach in building MOSAIC (**Mo**dular **S**ystem for **A**ssistive and **I**nteractive **C**ooking): a modular architecture for home robots that integrates multiple large-scale pre-trained models. In particular, we use large language models (LLMs) for interactive task planning, vision language models (VLMs) for visuomotor skills, and motion forecasting models for predicting human intents for collaboration. To the best of our knowledge, this is the first system to integrate multiple large-scale models in such a way that enables multiple home robots to collaborate with a human user to tackle complex, long-horizon tasks such as cooking.

While the principle of modularity has been central to developing robust real-world robotic systems (e.g. in autonomous driving), such systems often rely on meticulously engineered components. We introduce several key innovations to create an adaptive, scalable system that collaborates fluidly with humans. Our contributions can be organized into four groups:

1) **Interactive Task Planner.** We propose an architecture that embeds Large Language Models (LLMs) within a *behavior tree*. Prior work [1, 5–8] attempts to directly use LLMs for task planning. However, LLMs often make mistakes and are difficult to control. In response, we partition the action space and reasoning process as nodes in the tree, thereby reducing the complexity of reasoning required from the LLM and the overall error rate.
2) **Visuomotor Skills.** We propose a lightweight architecture that uses a pre-trained vision-language model for object identification and a policy learned via RL in simulation for action selection. In contrast to prior work [9–12] on completing tasks across an open vocabulary set of objects and varying environments, our method does not require *any* online demonstrations nor training large networks.
3) **Human Motion Forecasting.** We develop a method for forecasting human motion that allows robots to seamlessly collaborate with humans in manipulation tasks. Unlike prior works [13, 14] that model humans as static entities, we utilize large-scale human motion data [15] to train a forecasting model. Our approach focuses on generating human motion forecasts that optimize downstream planning performance, enabling our robots to plan safe and legible actions in close proximity to humans.
4) **Comprehensive Evaluation.** We conduct 60 end-to-end trials where two robots collaborate with a human user to cook complex, long-horizon recipes. We also extensively test individual modules with 180 episodes of visuomotor picking, 60 episodes of human motion forecasting, and 46 online user evaluations of the task planner. We run our system end-to-end with a real human user, completing 68.3% $(41/60)$ collaborative cooking trials of 6 different recipes with an average subtask completion rate of 91.6%.

II. Problem Statement

We focus on the task of a human user collaboratively cooking in a kitchen with two robots: one mobile manipulator and one tabletop manipulator. The user interacts with the system via natural language dialogue to decide on a recipe to cook. Once a recipe is determined, the system then allocates subtasks (e.g. fetching/putting away items, pouring, etc.) to the two robots and the human. The system replans subtasks based on feedback from the human and the status of the robots. Since the robots are assisting over the same tabletop, some of the subtasks involve collaborating closely with the human (e.g. handing over items or stirring while the human adds ingredients to the pot).

We make a set of simplifying assumptions in our work:

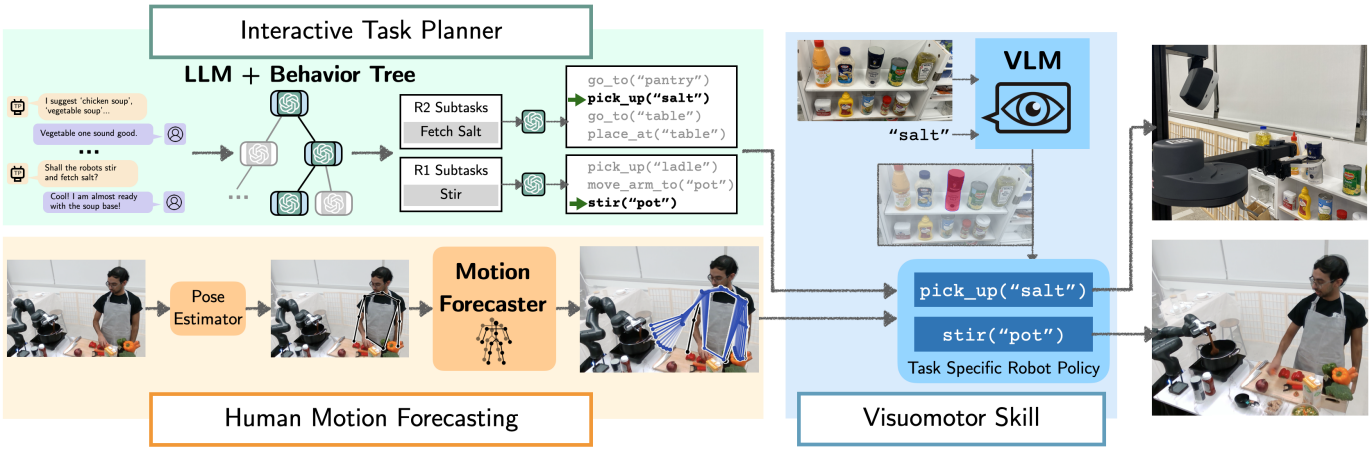


Fig. 2: **MOSAIC System Overview.** The *Interactive Task Planner* module communicates with the user via natural language to decide on a recipe. It assigns subtasks to each robot accordingly. The *Human Motion Forecasting* extracts and converts the human's 2D post to 3D coordinates, which it uses to predict future human motion. Simultaneously, a VLM takes image and language as input and produces a 3D grasp pose around the object of interest. Combined, all three are taken by the execution policy of the *Visuomotor Skill* module to produce a final robot action.

1) Access to a set of seed recipes: A recipe contains a set of subtasks with temporal dependencies. We seed the system with an initial set of recipes, but the user has the freedom to make modifications on the fly (e.g. adding an ingredient).
2) Access to a map: We assume that our system has mapped the kitchen ahead of time, so it is aware of where ingredients and tools are stored and how to navigate to different locations.
3) Full observability: We assume that objects are not occluded for detection and grasping, though they can be next to each other. We also assume that the upper torso of the human is visible to the cameras for tracking and prediction.
4) Skills API: We assume access to a library of robot skills that can be invoked with specific input parameters (e.g. `pick_up("salt")`, `stir()`).

III. Approach

We present MOSAIC, **Mo**dular **S**ystem for **A**ssistive and **I**nteractive **C**ooking, a modular architecture that combines multiple large-scale pre-trained models to solve collaborative cooking tasks. Fig. 2 shows an overview of MOSAIC. It consists of three main components: *1) Interactive Task Planner (III-A):* a module that interacts with real users via natural language to plan a diverse set of tasks and coordinate subtasks during the cooking process. *2) Visuomotor Skill (III-B):* a module that generalizes robot skills to a diverse set of kitchen objects and environments. *3) Human Motion Forecasting (III-C):* a module that leverages motion forecasting models to predict human motion, ensuring that robots can collaborate safely and fluidly with humans.

A. Interactive Task Planner

The goal of the task planner is to continuously interact with a human user using natural language, delegate subtasks to different robots or the user, and monitor progress. Concretely, the task planner interacts with the user to determine a task(e.g. "Prepare vegetable soup"). It represents the task $\mathcal{T}$ as a directed acyclic graph (DAG), which models temporal dependencies between different subtasks and determines available

Assign Subtask

Your goal is to assign subtasks to robots based on what the user has confirmed in the chat history...

You should choose a combination of these actions:
`assign(agent, subtasks), say(msg)`

For example, if the user has agreed to ..., you can choose:

```
assign("R1", ["fetch pepper"])
assign("R2", ["stir"])
say("Great, they will fetch pepper and stir the pot now.")
```

What to do when making recipe?

Your goal is to decide which node to go to based on the current situation in chat history...

You should choose ONLY ONE of these nodes:
ConfirmAssign, AssignSubtask, MarkAsComplete, Interrupt, NoOp

You must respect these constraints when making the decision:
- You must confirm your subtask assignment with the user before assigning...

For example, once the user agreed to your subtask proposal, you should choose:

`AssignSubtask`

Fig. 3: **Behavior node sippets.** Two prompt snippets of behavior nodes in our behavior tree. The top box shows a node that predicts a set of actions a_t^{high} to execute. The bottom box shows a node that predicts which child node n' to go to.

subtasks that can be assigned. The task planner also assigns and maintains a queue of subtasks for each robot. To execute a subtask (e.g. "fetch salt"), the task planner generates a code snippet that issues a series of API calls such as `go_to("pantry")`, `pick("pepper")`, etc. Please refer to the Appendix B for implementation details.

More formally, the task planner is a high-level policy π^{high} that takes as input the current high-level observation $o_t^{\text{high}} \in \mathcal{O}^{\text{high}}$, which contains chat history, current recipe, current robot queues, available subtasks, etc. It predicts one or more high-level actions $a_t^{\text{high}} \in \mathcal{A}^{\text{high}}$ such as `set_recipe(name)`, `assign(agent, subtasks)`, and `say(msg)`.

While many recent approaches [1, 5–8] directly use LLMs for task planning, we observe two main challenges. First, even

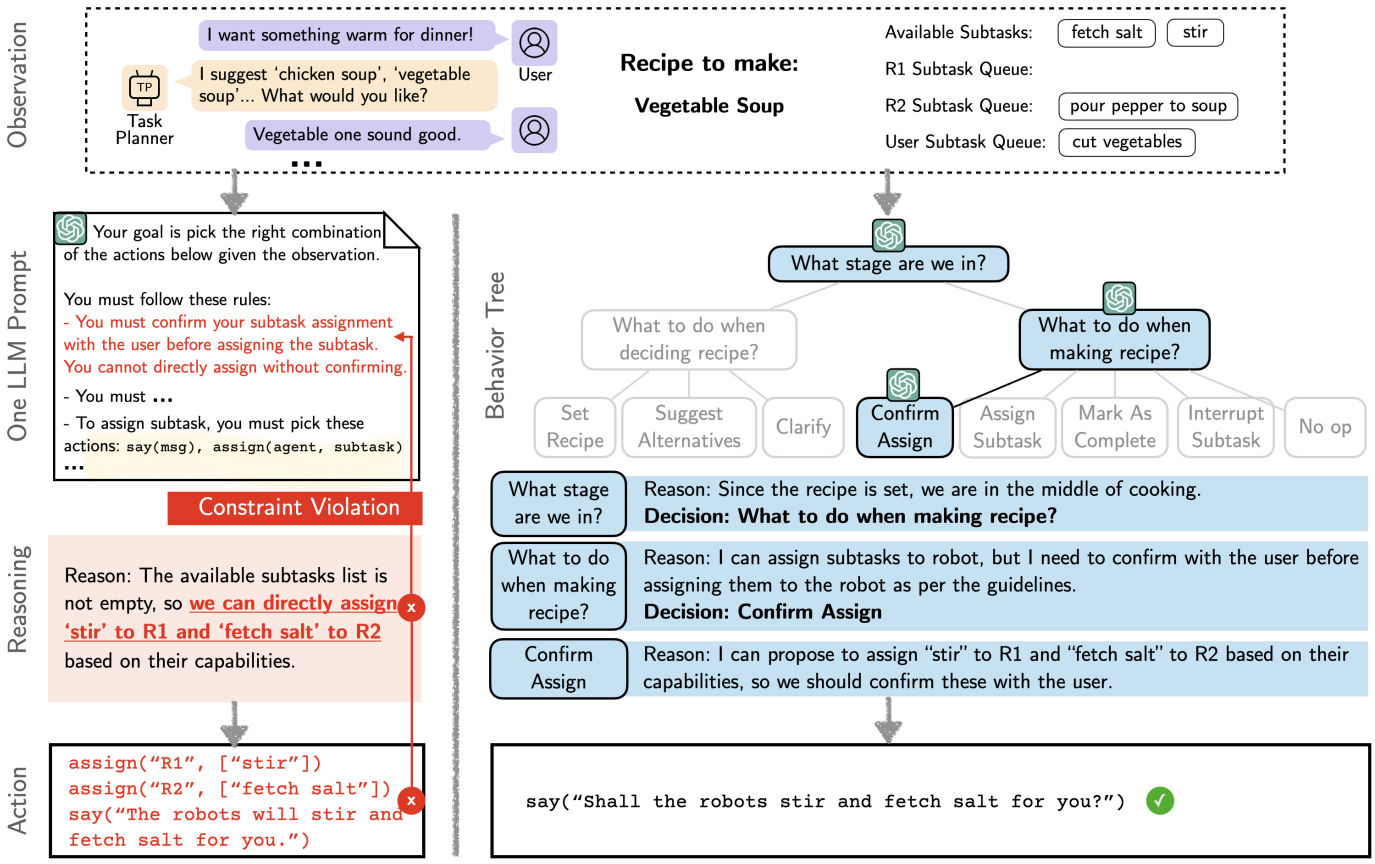


Fig. 4: **Tree-structured task planner vs single-prompt LLM.** We compare our approach against using one LLM prompt, which tends to violate constraints. Given the observation, the LLM with one monolithic prompt directly assigns subtasks to robots, which violates the constraint that it must confirm with the human before assigning tasks. Meanwhile, because our approach compartmentalizes the action space and reasoning process in a behavior tree, it is able to follow a correct reasoning path and correctly confirm its subtask proposal with the user.

with chain-of-thought prompting [16], since the action space is large and the reasoning process is complex, the LLMs make mistakes such as misinterpreting the observation or choosing incorrect actions. More importantly, the LLMs tend to violate safety constraints that the developer specifies, such as assigning subtasks without confirming with the user. Second, the developer has little control over the LLMs' behavior other than specifying the rules and constraints in one monolithic prompt, which is challenging to debug and scale.

To overcome both challenges, we propose an architecture that embeds LLMs within a behavior tree (BT) [17]. Each behavior partitions the action space and reasoning process, thereby reducing the complexity and potential error rate of the LLMs. Moreover, the modular nature of BT makes it easy to scale to multiple behaviors.

Embedding LLMs within Behavior Trees. A behavior tree is a hierarchical structure of individual behavior nodes, or simply behaviors. Each node n looks at the current observation o_t^{high} and chooses either a set of actions a_t^{high} to execute, or a child node n' to transition to. In our architecture, each node is a call to an LLM with a specific prompt and a pre-defined set of decisions to choose from.

For example, Fig. 3 shows snippets of prompts for different behaviors. The instructions describe the goal, the action space, which part of the observation to focus on, constraints to adhere to, and in-context examples. `Assign Subtask` is a leaf node that directly assigns subtasks to the robot and speaks to the user. On the other hand, the `What to do when making recipe?` behavior is a higher-level node that calls other behavior, e.g. `Confirm Assign`, `Assign Subtask`, etc.

Because each node's LLM is specialized to focus on a smaller reasoning problem, each behavior is empirically more reliable and easier to debug. Fig. 4 shows a comparison between the proposed approach and a baseline that uses a single monolithic prompt. The proposed approach more consistently respects the constraints the designer specifies (e.g. the requirement of confirming subtasks with the user before assigning them). These constraints are critical to ensure safe, predictable behaviors. Meanwhile, the LLM with a monolithic prompt is prone to violate such constraints in trying to follow all aspects of the instruction.

Finally, adding a new behavior is as simple as creating a prompt for that behavior and adding it as an option for other behaviors to invoke. No change to the code is necessary.

B. *Visuomotor Skills*

The goal of visuomotor skills is to execute subtasks assigned by the task planner. A skill is a low-level policy π that takes as input the current observation $o_t \in \mathcal{O}$ which consists of

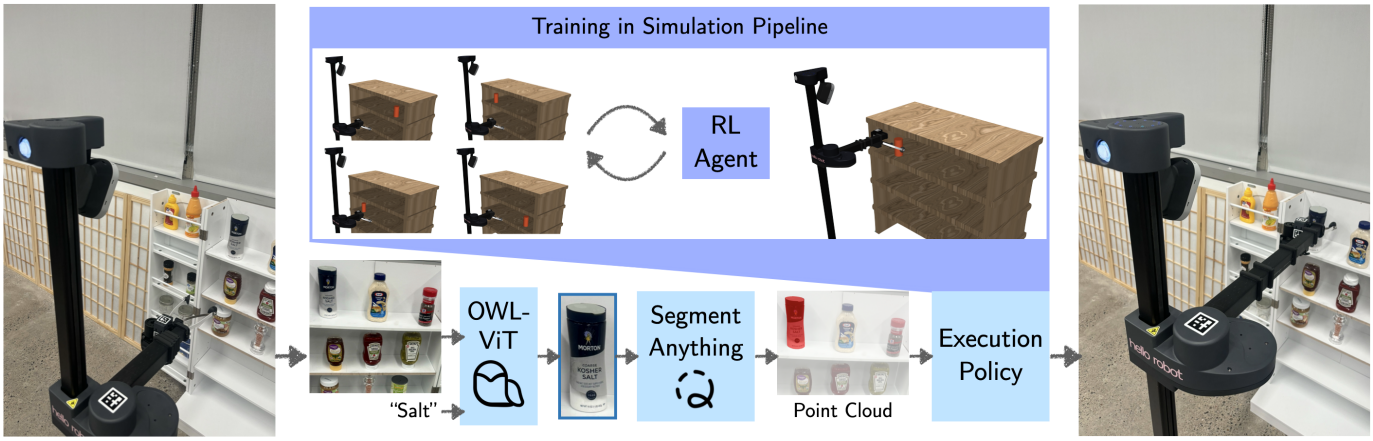


Fig. 5: **Architecture Overview for Visuomotor Skills.** We design a simulator that mimics the real environment on which the robot is rolled-out in. An RL agent is trained to predict actions given the goal position under a reward function that enforces environment constraints. At test time, the visuomotor module takes an image and natural language as input, and returns a bounding box around the object of interest using OwlViT [18]. This bounding box is passed into FastSAM [19], which segments out the image and back-projects it onto a point cloud to produce a 3D goal pose. This 3D goal pose is passed to the trained RL agent which produces the final action.

the current image and robot states, as well as the language instruction $\ell_t \in \mathcal{L}$ from the task planner. It predicts a low-level action $a_t \in \mathcal{A}$ which consists of gripper commands.

A common approach to train visuomotor skills is to imitate human demonstrations on a suite of tasks via end-to-end training [9–12, 20–22]. However, state-of-the-art methods using this approach generally require (1) good coverage of states and (2) expert action labels from those states. This includes data that shows the robot how to recover after making errors. Taken together, this leads to algorithms that require up to hundreds of hours of expert demonstrations, which is infeasible to collect.

Instead, we partition the end-to-end architecture into object-identification and action-execution modules. We offload object identification to pre-trained VLMs that can generalize to many objects, and we solve action execution by searching for a policy purely in simulation using reinforcement learning. In doing so, we have addressed both challenges without needing to collect any additional data. Figure 5 shows the architecture, which we discuss below.

Object detection via pre-trained models. Given our input image and language condition, we pass both through a pre-trained OwlViT [18] model, giving us a set of bounding boxes. To handle robot-specific viewpoints (that may be less common in the training data of these large VLMs), we filter the boxes using Non-Maximum Suppression and take the bounding-box coordinate with the highest CLIP similarity score [23]. We freeze the weights of both the OwlViT and CLIP models.

Grasp-pose generation via point-cloud segmentation. In the next phase of our pipeline, we use FastSAM [19] to obtain a more accurate segmentation of the object within the bounding box and back-project the segmented pixels through the depth camera's point cloud. We take our grasp-pose to be the center-of-mass of this projection.

Action prediction via model-based reinforcement learning. To predict the final actions, we design a simulator and reward function to train any general RL agent that takes as input some privileged information of the world, in this case the 3D grasp-pose, and outputs actions to reach that position without violating some set constraints. For a simple `pick()` policy, this constraint would be to not hit the pantry or knock over surrounding objects when reaching for the target object. This action module is trained entirely in simulation without any on-policy demonstrations and is applied directly during inference to predict per-timestep actions. Further implementation details can be found in the Appendix C. We currently use RL training in simulation for `pick()`, and use engineered policies for `move()` and `place()` since they are relatively straightforward to engineer.

C. Human Motion Forecasting

Safe and effective coordination with humans requires forecasting human motion and adapting robot plans accordingly. Accurate forecasts are critical for collaborative cooking, where robots work in close proximity to humans. For instance, observe the robot stirring a pot alongside a human partner in Figure 6. When the human moves in to put vegetables in the pot, the robot should anticipate that movement and make way for the human by retracting its arm back. *However, accurately forecasting human motion in dynamic environments such as kitchens is challenging.* Humans can perform a wide range of motions, such as manipulating various objects in the kitchen or moving between stations. Even with large amounts of training data and a long context window, current state-of-the-art models struggle to accurately predict human motion at all times. Instead, our goal is to build a forecasting model that generates predictions that sufficiently capture the impact of forecasted human motion during interactions with the robot.

Pre-training on Large-scale Data. We first pre-train our model on large-scale human activity data to generate smooth predictions of human motion given a history of joint positions as input. We use AMASS [15], a large dataset of human activity, encompassing over 300 subjects and 40 hours of motion capture data. The forecaster uses a short history of human motion as input and its predictions optimize the

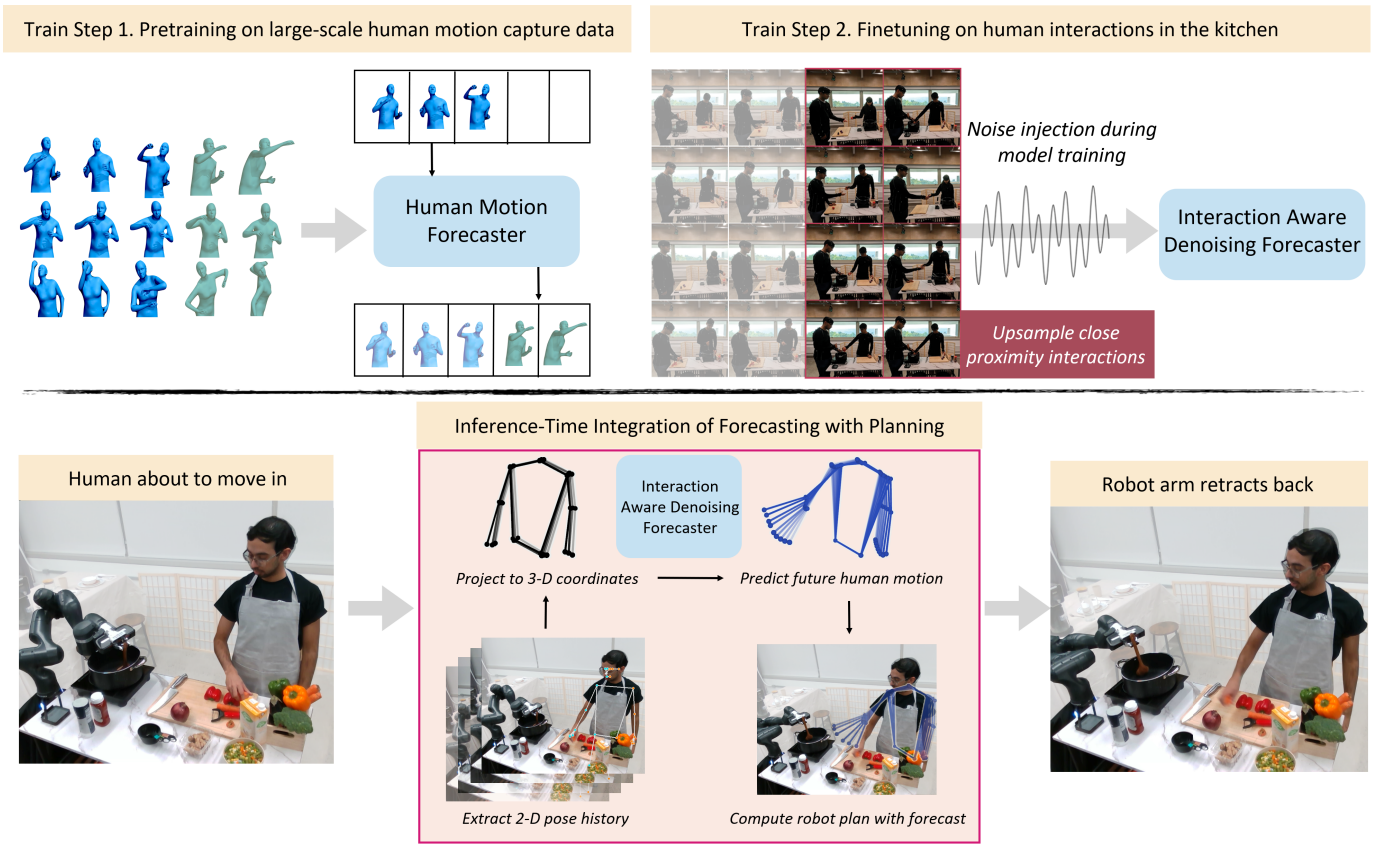


Fig. 6: **Human Motion Forecasting Overview**. (Top-Left) **Pre-training**. The forecaster is trained on a large dataset of human activities [15] to predict future motion given a short history of human poses. (Top-Right) **Interaction-Aware Fine-tuning**. The forecaster is then fine-tuned on CoMaD [24], a dataset of kitchen activities. At all steps of training, random noise is injected into the model's input to make the model robust to noise from camera inputs during inference. (Bottom) **Real-time, Vision-based Forecasting and Planning**. Given an RGB-D scene image, a pose detector extracts the human's 2D pose, which is converted to 3D coordinates using the camera's depth map. The motion forecaster predicts future human motion, which is used by the robot to plan actions.

likelihood of future human motion via MLE loss. After training on AMASS, the forecaster can produce dynamically consistent and reasonable human motion predictions. However, the motion predictions are not directly applicable to multi-agent kitchen settings. Firstly, the activities in AMASS consist of general human movements like jumping, walking, and dancing, which are not representative of daily kitchen activities. Secondly, the dataset consists of single-human motion, limiting the forecaster's ability to consider additional agents.

Fine-tune on Interaction Data. To ensure the forecaster's motion predictions are helpful for the robot to plan its actions, we utilize the Collaborative Manipulation Dataset (CoMaD) [24], a dataset consisting solely of human-human interactions in a kitchen setting. Episodes in CoMaD focus on a specific kitchen activity and contain short transition windows in which humans come into close contact with one another. Accurately forecasting human motion during these windows is critical to optimize the robot's planning performance. For each episode in CoMaD, we identify these transition windows and construct a *transition dataset*. We sample data equally from the *transition dataset* and the entire CoMaD dataset to train the motion forecaster. This approach helps the forecaster maximize task efficiency by upsampling critical periods of the interaction when the human is likely to approach and interact with the robot. Thus, instead of simply maximizing the likelihood estimate of future human motion across entire episodes, our forecaster accounts for the effect of its predictions on the robot's decision-making.

Inference Time: Real-time, Vision-based Forecasting and Planning. We represent the human pose using the 3D positions of 7 upper-body joints (shoulders, elbows, wrists, and neck). A single RGB-D camera aimed at the human's torso is used to detect their upper-body pose. The human joint locations are then identified on the RGB image using MediaPipe [25], a 2D pose detector. These locations are then back-projected to 3D world coordinates using the image depth map. Finally, the human poses are used to generate real-time motion forecasts that are used for robot planning. However, 3D coordinates obtained using the RGB-D camera's depth map are often noisy due to inaccuracies from the stereo camera. This poses a problem, as motion forecasting models trained on data from high-fidelity motion capture systems fail to make accurate predictions on out-of-distribution noisy inputs. By injecting random Gaussian noise into the model's input at training time, we force the forecaster to learn to denoise potentially noisy input and generate smooth forecasts.

Recipe	Typical Robot Skills Used	Success	Subtasks Comp.
Toss Salad	R1: Go-to, Pick, Go-to, Place R2: Pick, Stir	8 / 10	92.5%
Tuna Sandwich	R1: Go-to, Pick, Go-to, Place, Hand-over, Go-to, Place R2: Pick, Stir	8 / 10	96.0%
Vegetable Soup	R1: Go-to, Pick, Go-to, Place R2: Pick, Hand-over, Pick, Stir	8 / 10	96.0%
Corn Soup	R1: Go-to, Pick, Go-to, Place R2: Pick, Pour, Place, Pick, Stir	6 / 10	90.0%
Caesar Salad	R1: Go-to, Pick, Go-to, Place R2: Pick, Pour, Place, Pick, Stir	5 / 10	86.7%
Chicken Soup	R1: Go-to, Pick, Go-to, Place, Hand-over, Go-to, Place R2: Pick, Hand-over, Pick, Pour, Place, Pick, Stir	6 / 10	91.4%
		41 / 60	91.6%

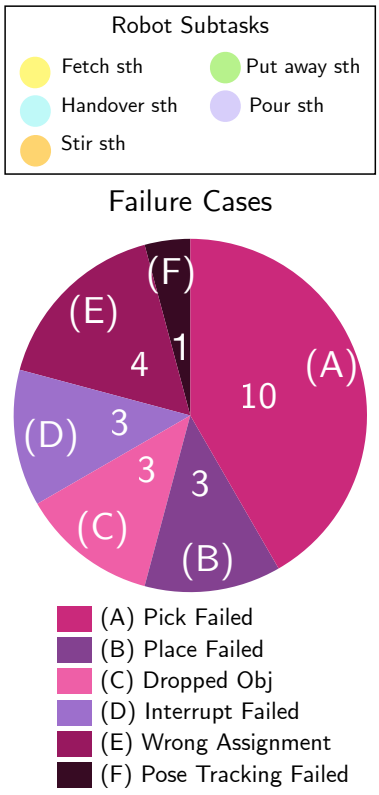


Fig. 7: **End-to-end results.** On-policy results for 6 recipes, where each recipe is tested through 10 trials. Each recipe contains various subtasks involving different robot skills. We report the number of trials that are completed without any errors and the individual subtask completion rate. We also categorize the failure cases. MOSAIC is able to complete 41/60 tasks with an average subtask completion rate of 91.6%.

IV. Experiments

We evaluate MOSAIC over 60 end-to-end trials where two robots collaborate with a human user to cook a combination of 6 recipes. We also conduct experiments to test individual modules that involve running online user studies, testing on unseen objects and backgrounds, and testing on multiple real users. In all experiments, the mobile manipulator is a 6-DoF Stretch Robot RE1 [26], and the tabletop manipulator is a 7-DoF Franka Emika Research 3 [27]. The kitchen also has two overhead RGB-D cameras that can perceive the workspace and capture a human's motion. To allow users to interact with the task planner, we use Google's speech-to-text APIs [28] to transcribe user's verbal instructions and its text-to-speech APIs to vocalize the task planner's responses.

A. End-to-end Trials

We conduct a total of 60 end-to-end trials with two robots and a user collaboratively making 6 recipes. Figure 7 shows a table with the different recipes (tasks), the different subtasks, and the robot skills involved. Each recipe involves a different combination of robot skills and different types of interaction with the user. For example, users provide vague instructions, interrupt a robot's subtask, and add new subtasks that are not in the recipe. For each trial, we compute two metrics: was the trial successful, and what was the subtask completion rate.

Overall, MOSAIC completes 41/60 (68.3%) collaborative cooking trials of 6 different recipes with an average subtask completion rate of 91.6%. We analyze two specific questions:

How does MOSAIC scale with longer horizon tasks? We test a range of recipes, from "Toss Salad", which involves 6 skills, to "Chicken Soup", which involves 14 skills. While MOSAIC's success rate drops with the increasing horizon as one would expect, it does not fall off exponentially and stays above 50%. A key reason is that each module in MOSAIC is trained to be robust to errors in incoming input (e.g. the task planner handles delays made by a robot, the visuomotor skills `pick()` handles errors from `go_to()`, the forecasting handles errors from pose estimation, and so on).

Does modularity help localize failures to specific modules? As each module has sub-modules, each with a clear input/output contract, localizing an error is easily automated. We use this to cluster failures into the following 5 categories, shown also in Figure 7:

(A) *[Visuomotor Skill] Failed to pick up the object:* Sometimes, the VLM selects an incorrect object given the object prompt, which we further analyze in section IV-B. Other times, the predicted goal has a high error, causing the gripper to miss the object.

(B) *[Visuomotor Skill] Failed to successfully place the object:* Errors in the `go_to()` skill leave the robot too far away from the table to successfully place an object. Sometimes, the robot releases an object from an incorrect height, causing it to topple.

(C) *[Visuomotor Skill] Dropped the object during a skill:* The

Fig. 8: **Vision backbone example failure cases.** We demonstrate three regimes where the VLM module fails: high-density images with many objects, backgrounds with low contrast to the object of interest, and prompt sensitivity.

`stir()` and `pour()` skill may drop an object due to an insufficiently stable grip.

(D) *[Interactive Task Planner] Failed to interrupt a subtask:* When the user asks the robot to stop their current subtask, the speech-to-text module sometimes fails to correctly transcribe user's short command. The unclear transcription causes the task planner to ask the user for clarification instead of immediately interrupting the robot.

(E) *[Interactive Task Planner] Assigned an incorrect subtask:* The task planner misunderstands the user's command and re-assigns a completed subtask to the robot.

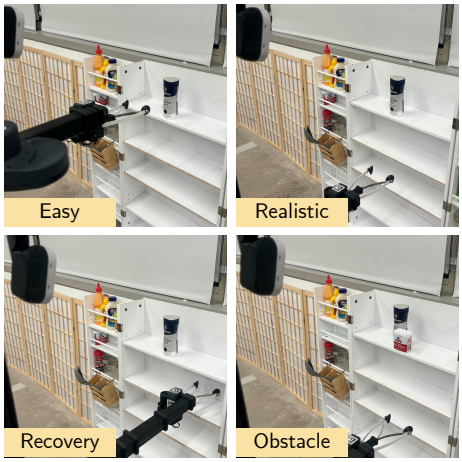
(F) *[Human Motion Forecasting] Pose Tracking Failed:* The human's pose moved outside the camera's view, causing a tracking error while forecasting motion.

B. Visuomotor Skills

	Single Object	Medium Density	High Density
OwlVit + CLIP (Ours)	10/10	10/10	6/10
OwlVit only	10/10	3/10	0/10

TABLE I: **On-policy Evaluations of Visuomotor Architecture.** The architecture is tested on its ability to pick up the language-specified object when (1) a single object is in the pantry, (2) 2-6 objects are in the pantry, (3) 7-15 objects in the pantry. We observe that using CLIP post-processing is critical in allowing for stable and correct bounding box predictions as the number of objects in an image increases.

While the end-to-end trials demonstrate the performance of the visuomotor skill as a whole, we now analyze the internal modules independently. We examine the vision-language module and the policy module separately, and aim to qualitatively and quantitatively answer the following questions:



	Easy	Realistic	Recovery	Obstacle
RL in Simulation	10/10	10/10	10/10	4/10
IK	10/10	3/10	0/10	2/10
BC	6/10	4/10	0/10	3/10

TABLE II: **On-policy Evaluations of Policy Module.** We see the RL agent trained in simulation successfully reaches the goal without hitting the pantry, despite an adversarial reset placements such as the arm extended into a lower pantry level. However, success rate deteriorates as object placements violate the initial assumptions made about the simulator used to train the agent.

1) **How well does an off-the-shelf VLM perform as a vision backbone?** We compare an off-the-shelf OwlViT [18] with a CLIP post-processor [23] against an OwlViT-only vision module for completing the `pick()` policy. We test the limitations and failure cases by varying both the density and the type of objects present in the image.
2) **Does training an RL agent purely in simulation produce a desirable policy?** We evaluate our policy module against simple inverse kinematics and behavior cloning. Across 4 different initial positions, we test each policy's ability to complete the `pick()` without violating environment constraints.

Vision-Language Module. We first quantitatively evaluate the components necessary for a vision-language module to reliably identify objects in the pantry. Since OwlViT is trained to be an open-vocabulary *detector* and calculates loss over the object proposals within an image [18], we suspect it is less adept at language-conditioned classification than the purely contrastively-trained CLIP [23]. To this end, we compare our approach of predicting 10 boxes and selecting the highest one using CLIP against the baseline where OwlViT directly predicts the final bounding box.

In Table I, we present the `pick()` policy success rates of using these two backbones across three increasingly clustered pantries. We find that as the number of objects in the pantry increases, OwlViT's success rate drops significantly. Notably, OwlViT suffers less from placing *a* bounding box over the correct object, but rather more from confidently classifying

Task →	Reactive Stirring			Robot to Human Handovers	
Model ↓	Safety Dist. (cm) ↑	Time to React (ms) ↓	Collisions ↓	Time to Goal (s) ↓	Path Length (cm) ↓
Current	13.5 (±0.2)	135.4 (±10.4)	2/10	1.54 (±0.1)	31.5 (±1.2)
Forecast (Base)	19.9 (±0.2)	64.9 (±9.8)	0/10	1.67 (±0.2)	32.7 (±3.0)
Forecast (Ours)	23.1 (±0.2)	48.9 (±5.0)	0/10	1.15 (±0.1)	22.4 (±0.2)

TABLE III: **Task-Specific Performance Metrics.** We evaluate the robot's interactions with the human user on 2 collaborative manipulation tasks. Integrating forecasts into the robot's skills improves safety and fluidity across all metrics. We observe that solely relying on the current human pose during Reactive Stirring is risky and causes human-robot collisions. Robot-Handover tasks are completed more efficiently using forecasted human positions.

Fig. 9: **On-Policy Reactive Stirring.** (Left) **Current**: Using the human's current pose results in a delayed robot reaction and a collision once the human's hand enters the pot. (Right) **Forecast** Using the forecasted human position results in a smoother interaction and quicker reaction time, avoiding a collision.

that bounding box as the correct object of interest. Instead, CLIP takes all bounding box proposals in isolation to predict the one most aligned with the language embedding, leading to more stable predictions and, thus, a more reliable grasp-pose for our policy module.

While the above shows the importance of CLIP post-processing to reliable goal predictions, Figure 8 qualifies three failure regimes of our vision module. First, the presence of too many objects (especially similarly-shaped ones such as various seasoning bottles) leads to a suboptimal set of bounding box proposals for CLIP to score. Furthermore, when lighting and/or color blends the object contours into the background, only parts of the object may be included in the bounding box, thus resulting in a lower CLIP score. Finally, we find that more specific prompts produce better bounding box proposals.

Policy Module. We evaluate the policy module for different scenarios where the end effector must grasp the object a) *Easy*: when the gripper is close to the object, b) *Realistic*: when the gripper is retracted, c) *Recovery*: when the gripper is in an extended position away from the object, and d) *Obstacle*: when the object is partially occluded. We compare with a simple inverse kinematics (IK) baseline and a behavior cloning (BC) policy. Additional details can be found in the Appendix C. In all cases, we consider the task failed if the robot hits the pantry or an object is knocked over. We present our results in Table II. IK generally fails to avoid the pantry when reset to a lower position. BC fails to recover when the base moves or is reset to a state outside its training distribution. Since our RL agent is trained in simulation, it provides a robust policy that is both constraint-aware and has good coverage. Nevertheless, training in simulation requires a model and specified reward function, and thus is susceptible to failure when provided a scenario not captured by the simulator or reward. In these cases, we posit BC lends itself to more expressive policies if provided with sufficient demonstration.

C. Human Motion Forecasting

We now analyze the human-motion forecasting module through on-policy evaluation of a 7-DOF Franka robot arm collaborating with a real human user. We evaluate two specific skills, `stir()` and `handover()`, that rely on forecasts of the human user and aim to answer the following questions:

1) **How much does forecasting help over simply using the current pose for downstream performance?** We run the robot's `stir()` and `handover()` skills on-policy with a human user and compare task-specific metrics.
2) **Is our method robust to noisy human detection from vision-based sensors?** We evaluate our method on collaborative manipulation tasks requiring varying degrees of precision, comparing our forecaster with a baseline approach that does not denoise its model inputs.

For each skill, we compute specific metrics to measure on-policy performance. We compare against two baselines: (1) *Current* which assumes the current human pose will be its pose across the entire planning horizon, and (2) *Forecast (Base)* that predicts a forecast without de-noising. Each baseline and skill combination is evaluated 10 times for a total of 60 evaluations.

Reactive Stirring. We evaluate the `stir()` skill in the following setting: the robot stirs the pot while the human chops vegetables and periodically adds items to the pot. Interactions occur only when the human moves their hand over the pot. We measure Time to React (ms), the time it takes for the robot to detect that the human arm is reaching in for the pot after they pick up vegetables, as well as Safety Dist (cm), the minimum distance maintained between robot and human hand during interactions. We also measure the number

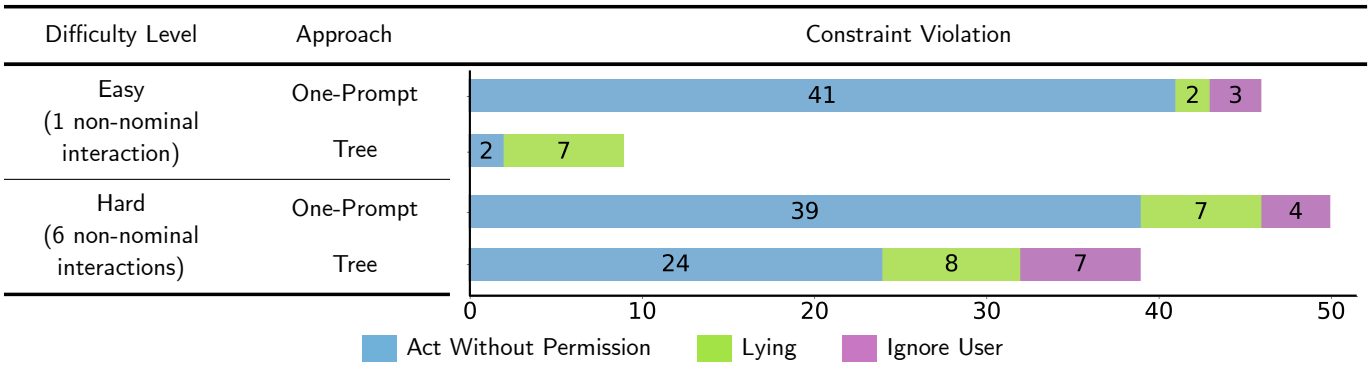


Fig. 10: **Task Planner Constraint Violations in Integration Tests.** Each approach is evaluated on 5 random unique recipes from beginning to end with varying numbers of non-nominal interactions. Each approach gets run 3 times per recipe. We present the total number of constraint violations across all runs for each difficulty level. *Tree* has the lowest total number of constraint violations compared to *One-Prompt* for all difficulty levels.

of COLLISIONS, cases when the human hand comes too close (within a threshold) to the robot arm. (In such cases, the robot's emergency stop is triggered.) We observe that the robot reacts very late while only using the human's current pose, and this leads to a collision in two cases, as shown in Table III. In comparison, using forecasts significantly improves on all metrics and, more importantly, avoids collisions. Finally, while training forecasts without denoising show improvement in task metrics over using the current human pose, there is greater variability in its performance (measured by the variance of each metric).

Robot to Human Handover. In this task, the user asks the robot to pick up and handover objects. We evaluate the speed and efficiency of the robot's `handover()` skill after it has picked up the requested object. To measure speed, we measure the average TIME TO GOAL (MS), the time taken by the robot to bring the object to the human and complete the handover. The efficiency of the robot's movement is measured through its PATH LENGTH (CM), which measures the distance tracked by the robot's end-effector while moving towards the human's hand. Using the *Current* human pose for handovers, the robot simply follows the current human wrist position. Table III shows that the robot is significantly slower in completing the task using this approach compared to using the handover location predicted by our forecaster. Using the human's forecast, the robot moves directly toward the final handover location, finishing the skill with significantly shorter trajectories than following the current pose. For this skill, we note that using forecasts that have not been denoised does not improve planning metrics. The 3D human pose detected by the vision-based system is often noisy, and the predicted forecasts can be erratic in these cases, leading to jerky movements by the robot arm. This shows the importance of our denoising module to ensure safety during close proximity collaboration.

D. Interactive Task Planner

To test the task planner's robustness, we conduct (1) an integration experiment to test how the system handles increasing amount of complex interaction, and (2) an online user study to analyze how the system interacts with real users. Concretely, we compare the task planner that embeds LLMs within a behavior tree (*Tree*, shown in Figure 4) against using one LLM prompt (*One-Prompt*). For this baseline, its monolithic LLM prompt has constraints that the task planner must follow, explanations of what actions to choose in each situation, and in-context examples. With the two experiments, we seek to answer the following questions:

1) **How well can the task planner robustly handle complex interactions with a user?** In both experiments, the user is asked to interfere with the task planner's normal workflow. Integration test contains test cases to examine whether an approach is able to properly respond to each type of non-nominal interaction.
2) **How does the LLM with behavior tree architecture help enforce the task planner to respect the constraints?** We analyze the number of constraint violations that each approach has as the interaction complexity increases and when it interacts with a real user.

Difficulty	Approach	Avg. Non-nominal Pass Rate (%)
Easy	One-Prompt	100 ± 0.00
	Tree	90.0 ± 5.35
Hard	One-Prompt	60.0 ± 9.04
	Tree	94.0 ± 2.30

TABLE IV: **Task Planner Success Rate in Integration Tests.** We present the average percentage of non-nominal interaction that gets successfully handled by the task planner. *Tree* can more robustly handle complex interactions compared to *One-Prompt*.

Integration Test. To systematically test the task planner, we design unit tests of what the task planner should do given a type of user interaction. In addition to nominal interactions, where the user gives clear instructions and agrees with the task planner's proposal, we identify 4 non-nominal interaction modes. For example, when the user is making a recipe, they might modify the task planner's subtask assignment proposals or request the robots to do subtasks outside of the recipe. To generate natural interactions during the tests, we create an LLM prompt that mimics an everyday user who provides

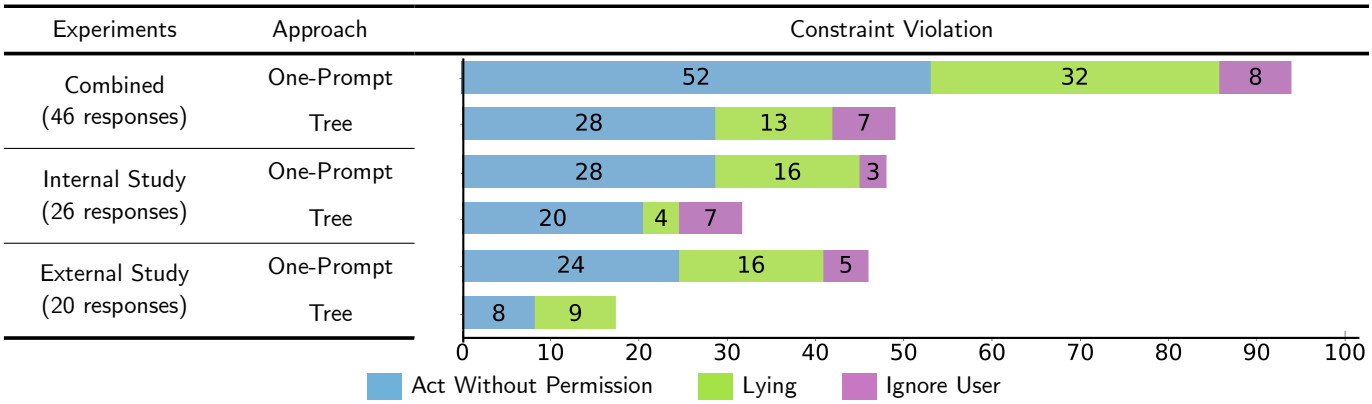


Fig. 11: **Task Planner Constraint Violations During Real User Interactions.** We receive 46 responses in total (26 from internal and 20 from external study). Each user gets assigned either *One-Prompt* or *Tree* to interact and make a random recipe together. We present the total number of constraint violations per variant. *Tree* exhibits less constraint violations compared to *One-Prompt* in such real-user interactions.

different instructions based on the interaction mode we set programmatically.

We create the following categories with increasingly more complex interactions: "Easy" with only one random non-nominal interaction and "Hard" with six. For each difficulty level, we test the approaches on 5 recipes, and for each recipe, we run the entire cooking process 3 times with the same set of non-nominal interactions. This experiment results in 30 runs per approach and an average of 34 chats per run.

We measure the average percentage of unit tests passed and analyze the number of times that the task planner has violated the constraints specified in the prompts. The constraint violations are grouped into 3 categories.

(A) Act Without Permission: The task planner assigns or removes subtasks without the user's permission.

(B) Lying: It claims robots can do subtasks beyond their capabilities, or it claims to do something but does not.

(C) Ignore User: It does not respond to the user's instruction.

Table IV shows that, overall, compared to *One-Prompt*, *Tree* has a higher percentage of average unit tests passed. Meanwhile, Figure 10 highlights that *Tree* consistently violates fewer constraints. These results suggest that compartmentalizing the action space and simplifying each LLM's reasoning problem significantly helps the task planner to respect constraints.

Both approaches struggle with the constraint that it must not act without getting permission from the user first, but how each approach violates this is different. Even when cooking has just started, *One-Prompt* tends to directly assign subtasks to the robots without confirming with the user. Meanwhile, *Tree* only tends to violate this constraint when it directly assigns follow-up subtasks to the robots in response to the user saying that they have completed their subtasks.

User Study. To verify that the task planner still respects the constraints when interacting with real users, we conduct an online user study, where a user talks to the task planner via a web-based chat. A user gets randomly assigned the proposed approach (*Tree*) or the baseline (*One-Prompt*). Then, the user gets randomly assigned a recipe to complete, and for each recipe, there is at least one response per approach. We require each user to complete 3 distinct non-nominal interaction modes during the cooking process before completing a survey to evaluate if the task planner has violated any constraints. We received 46 responses in total, where 26 responses are from lab members who are not familiar with the task planner's capabilities, and 20 are from external users on a crowd-sourcing website called Prolific [29]. Figure 11 shows that when interacting with a real user, *Tree* consistently violates fewer constraints compared to *One-Prompt*.

With 23 participants in each approach, we see a reduction in violating the constraint "Act Without Permission" in *Tree* (1.22 ± 0.25) compared to *One-Prompt* (2.26 ± 0.42) with a statistical significance of $p_{.041}$. *Tree* also lies less often to the user (0.56 ± 0.24 vs 1.39 ± 0.31), with a statistical significance of $p_{.040}$. While there was no significant effect in number of times the user was ignored, $p_{.836}$, these violations are few and far between in both approaches (0.30 ± 0.15 vs 0.34 ± 0.15). All reported errors are standard errors.

Users are asked to leave feedback about their experience, which provides more insights into these findings. Generally, users assigned with *Tree* have a better experience than with *One-Prompt*. One user assigned with *One-Prompt* commented "I could definitely see myself blowing my top with the level of disobedience." while another user assigned with *Tree* commented "It worked as expected, quick and concise answers, compliant, didn't make any mistakes". Another user assigned with *Tree* commented "I felt like it was a little overbearing at times, trying to assign many tasks at once," which suggests that although the tree does not assign subtasks without the user's permission, this behavior may be annoying for some users.

V. Related Works

Home Robots. Recent research efforts have attempted to provide robots with generalist capabilities to sufficiently adapt to home-like environments [1–4, 30]. However, many of these works [3, 4] are limited to completing simple predefined tasks that don't require explicit task planning, e.g. picking a single

item. Liu et al. [30] approaches open-vocabulary navigation and manipulation similar to our work, yet still sidesteps the challenge of a dynamic environment by assuming a static representation of the world after initialization. On the other hand, some works consider multi-arm/multi-robot planning for collaborative tasks [8, 31–33]. For example, Mandi et al. [8] attempts to extend its multi-robot framework to the human-robot setting, yet significantly constrains human-robot collaboration by forcing the human to complete a specific task before the robot can go ahead with its own task. In this paper, we aim to overcome these limitations by designing a system that enables multiple robots to fluidly collaborate alongside humans to perform a wide range of tasks.

Task Planning. A task planner takes as input a high-level task, e.g. cooking a recipe, and generates a plan, e.g. a sequence of sub-tasks, to achieve that goal. Traditional approaches frame this as a search problem and invoke a symbolic planner to solve it [34–37]. However, using these methods for everyday tasks is challenging because they require pre-defining the search space and lack a natural-language interface to interactively communicate the task. Recent work leverages LLMs for task planning to overcome both of these limitations. In single-robot settings, given a clearly defined language goal, recent work can be categorized as generating a list of actions as the plan [38–41], synthesizing code that calls robot action API [5, 42–44], or translating to a problem solvable by a classical planner [6, 45]. However, none of these systems interact with humans and coordinate tasks for both humans and robots. Li et al. [7] has the closest task planning framework to our approach, where the LLM takes a specific natural language goal to generate a step-by-step plan before synthesizing robot code for each step. However, because we are solving a multi-agent task planning problem involving two robots and a user, our task planner cannot simply output a list of steps. It must continuously communicate with the user to properly allocate subtasks to suitable agents.

Visuomotor Skills. Several recent works study the application of pre-trained vision-language models (VLMs) to robotics [10–12, 21, 46–49]. One family of recent work [10–12, 21, 48, 49] integrate pre-trained VLMs in an end-to-end fashion, e.g. segmenting out regions of interest to assist in action prediction [12, 49]. A second flavor of approach [1, 40, 47, 50] leverages VLMs to recognize affordances and constraints in the environment and provide corresponding execution instructions through language [47] or code [50]. Our model is similar in this aspect, where we distinguish the training objectives of environment perception and action execution. This effectively liberates us from needing a large dataset of humans or robots demonstrations to provide good coverage [10–12, 21, 22] and from having to worry about embodiment mismatch between large-scale robot learning datasets [51, 52]. However, in contrast to prior work [44, 47, 50, 53], we train our action policy using reinforcement learning in simulation where affordances are provided by the VLM and constraints are inherent to the simulator.

Human Motion Forecasting. Collaborative manipulation tasks in close proximity to humans require predicting human motion. This is a challenging problem since human motion is complex and highly variable. A common approach is to sidestep the problem of motion forecasting [13, 14] by considering the human to be static. Instead, recent research is moving towards the use of neural networks and supervised learning to predict future human motion based on a short history of past joint positions [54–57]. The release of large open-sourced datasets of human motion [15, 58] has made it possible to train large RNN and GNN-based neural network models for human-pose forecasting [59, 60]. Consequently, these datasets have been integrated into robot motion planning, focusing on collaborative manipulation tasks [24, 61]. Closest to this work, ManiCast [61] proposed a framework to learn cost-aware human forecasts. However, this approach relies on a bulky motion camera setup, requiring the user to wear a motion capture suit with markers. In this work, we run our integrated human motion forecasting and planning system in real-time using a single RGB-D camera to track human pose.

VI. Discussion and Limitations

In this paper, we present a modular system capable of controlling two robots to interactively cook a variety of recipes with a human user. Leveraging an ensemble of large-scale, pre-trained models, our system communicates with the user, forecasts their intents, and completes a series of visuomotor skills. To validate our design decisions, we conduct extensive experiments in the real world with multiple human users. By adopting a modular approach, we conduct detailed evaluations of its components in isolation. Furthermore, this process of modular evaluation has been instrumental in uncovering potential failure modes. In this section, we discuss the key limitations of this work that we plan to tackle in the future:

1. Grounding the task planner. The task planner can only observe the user's state through conversation. Grounding the task planner with pre-trained models to track the workspace states and predict the user's higher-level intent is an interesting direction for future work.

2. Scaling to more complex skills and environments. We show our robots performing a number of skills like picking, placing, and moving. The extension to more intricate tasks such as cutting, rolling, and spreading is less trivial. While our system is tested extensively on different recipes, our experiments are restricted to one kitchen environment. Future work will attempt to measure the limits of our system's generalization capability in the face of a wider range of kitchen environments.

3. Learning from interactions and feedback. The system's capabilities remain static after being deployed in an everyday user's household. An exciting area of future research is to continuously learn from real-time human feedback and interactions. A natural next step for our system is to use its history of collaborating with the user to adapt to the user's preferences.

4. Error detection and recovery. While the human could describe errors of the system to the high-level task planner using natural language, autonomously detecting these errors

in a closed-loop fashion would increase the success rate and improve the user experience.

5. Frozen pre-trained models. Several components of our system leverage multiple pre-trained models (e.g. the visuomotor skill vision backbone) to improve data efficiency and performance. Nevertheless, in the absence of finetuning, we are still generally limited to the innate capabilities of each model. For example, there may exist objects in our kitchen that are outside of the training distribution of such models. An ideal system would adapt gracefully to these new scenarios.

We architect a set of modular frameworks that utilizes large-scale, pre-trained models to quickly equip multi-agent systems with generalizable skills. These characteristics make MOSAIC a desirable foundation for collaborative human-robot systems in complex home environments and for future work that further refine and expand this system's capability.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Max Bajracharya, James Borders, Richard Cheng, Daniel M. Helmick, Lukas Kaul, Dan Kruse, John Leichty, Jeremy Ma, Carolyn Matl, Frank Michel, Chavdar Papazov, Josh Petersen, Krishna Shankar, and Mark Tjersland. Demonstrating mobile manipulation in the wild: A metrics-driven approach. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi: 10.15607/RSS.2023.XIX.055. URL https://doi.org/10.15607/RSS.2023.XIX.055.

[3] Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[4] Sriram Yenamandra, Arun Ramachandran, Karmesh Yadav, Austin Wang, Mukul Khanna, Theophile Gervet, Tsung-Yen Yang, Vidhi Jain, Alexander William Clegg, John Turner, et al. Homerobot: Open-vocabulary mobile manipulation. *arXiv preprint arXiv:2306.11565*, 2023.

[5] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2023.

[6] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

[7] Boyi Li, Philipp Wu, Pieter Abbeel, and Jitendra Malik. Interactive task planning with language models, 2023.

[8] Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models, 2023.

[9] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[10] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[11] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[12] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, et al. Open-world object manipulation using pre-trained vision-language models. *arXiv preprint arXiv:2303.00905*, 2023.

[13] Wei Yang, Balakumar Sundaralingam, Chris Paxton, Iretiayo Akinola, Yu-Wei Chao, Maya Cakmak, and Dieter Fox. Model predictive control for fluid human-to-robot handovers. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6956–6962. IEEE, 2022.

[14] Emrah Akin Sisbot and Rachid Alami. A human-aware manipulation planner. *IEEE Transactions on Robotics*, 28(5):1045–1057, 2012.

[15] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, October 2019.

[16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[17] Michele Colledanchise and Petter Ögren. Behavior trees in robotics and AI: an introduction. *CoRR*, abs/1709.00084, 2017. URL http://arxiv.org/abs/1709.00084.

[18] Georg Heigold, Matthias Minderer, Alexey Gritsenko, Alex Bewley, Daniel Keysers, Mario Lučić, Fisher Yu, and Thomas Kipf. Video owl-vit: Temporally-consistent open-world localization in video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13802–13811, 2023.

[19] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023.

[20] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action

diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[21] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[22] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[23] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[24] K. Kedia, Atiksh Bhardwaj, Prithwish Dan, and Sanjiban Choudhury. Interact: Transformer models for human intent prediction conditioned on robot actions. *ArXiv*, abs/2311.12943, 2023.

[25] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Lixuan Zhu, Fan Zhang, and Matthias Grundmann. Blazepose: On-device real-time body pose tracking. *ArXiv*, abs/2006.10204, 2020.

[26] Charles C. Kemp, Aaron Edsinger, Henry M. Clever, and Blaine Matulevich. The design of stretch: A compact, lightweight mobile manipulator for indoor human environments, 2022.

[27] Franka research 3, 2022. URL https://franka.de/documents.

[28] URL https://cloud.google.com/speech-to-text/.

[29] Prolific, 2014. URL https://www.prolific.com.

[30] Peiqi Liu, Yaswanth Orru, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.

[31] Mehmet Dogar, Andrew Spielberg, Stuart Baker, and Daniela Rus. Multi-robot grasp planning for sequential assembly operations. *Autonomous Robots*, 43:649–664, 2019.

[32] Huy Ha, Jingxi Xu, and Shuran Song. Learning a decentralized multi-arm motion planner. *arXiv preprint arXiv:2011.02608*, 2020.

[33] Argtim Tika and Naim Bajcinca. Predictive control of cooperative robots sharing common workspace. *IEEE Transactions on Control Systems Technology*, 2023.

[34] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, dec 2011. ISSN 0001-0782. doi: 10.1145/2043174.2043195. URL https://doi.org/10.1145/2043174.2043195.

[35] Yuqian Jiang, Shiqi Zhang, Piyush Khandelwal, and Peter Stone. An empirical comparison of pddl-based and asp-based task planners. *CoRR*, abs/1804.08229, 2018. URL http://arxiv.org/abs/1804.08229.

[36] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(02)00186-8. URL https://www.sciencedirect.com/science/article/pii/S0004370202001868. Knowledge Representation and Logic Programming.

[37] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *CoRR*, abs/1106.4561, 2011. URL http://arxiv.org/abs/1106.4561.

[38] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

[39] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.

[40] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.

[41] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: from natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, November 2023. ISSN 1573-7527. doi: 10.1007/s10514-023-10131-7. URL http://dx.doi.org/10.1007/s10514-023-10131-7.

[42] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022.

[43] Huaxiaoyue Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, and Sanjiban Choudhury. Demo2code: From summarizing demonstrations to synthesizing code via extended chain-of-thought, 2023.

[44] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, November 2023. ISSN 1573-7527. doi: 10.1007/s10514-023-10139-z. URL http://dx.doi.org/10.1007/s10514-023-10139-z.

[45] A. Mavrogiannis, Christoforos Mavrogiannis, and Yian-

nis Aloimonos. Cook2ltl: Translating cooking recipes to ltl formulae using large language models. *ArXiv*, abs/2310.00163, 2023.

[46] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. *arXiv preprint arXiv:2203.10421*, 3(4): 7, 2022.

[47] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palme: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[48] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv*, 2022.

[49] Siddharth Karamcheti, Suraj Nair, Annie S Chen, Thomas Kollar, Chelsea Finn, Dorsa Sadigh, and Percy Liang. Language-driven representation learning for robotics. *arXiv preprint arXiv:2302.12766*, 2023.

[50] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[51] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[52] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*, 2019.

[53] Priya Sundaresan, Suneel Belkhale, Dorsa Sadigh, and Jeannette Bohg. Kite: Keypoint-conditioned policies for semantic manipulation. *arXiv preprint arXiv:2306.16605*, 2023.

[54] Hejing Ling, Guoliang Liu, Liujuan Zhu, Bin Huang, Fei Lu, Hao Wu, Guohui Tian, and Ze Ji. Motion planning combines human motion prediction for human-robot cooperation. In *2022 12th International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 672–677. IEEE, 2022.

[55] Vaibhav Unhelkar, Przemyslaw A. Lasota, Quirin Tyroller, Rares-Darius Buhai, Laurie Marceau, Barbara Deml, and Julie A. Shah. Human-aware robotic assistant for collaborative assembly: Integrating human motion prediction with planning in time. *IEEE Robotics and Automation Letters*, 3:2394–2401, 2018.

[56] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 885–892, 2015.

[57] Vignesh Prasad, Dorothea Koert, Ruth Maria Stock-Homburg, Jan Peters, and Georgia Chalvatzaki. Mild: Multimodal interactive latent dynamics for learning human-robot interaction. *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 472–479, 2022.

[58] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013.

[59] Wei Mao, Miaomiao Liu, and Mathieu Salzmann. History repeats itself: Human motion prediction via motion attention. In *European Conference on Computer Vision*, 2020.

[60] Theodoros Sofianos, Alessio Sampieri, Luca Franco, and Fabio Galasso. Space-time-separable graph convolutional network for pose forecasting. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11189–11198, 2021.

[61] K. Kedia, Prithwish Dan, Atiksh Bhardwaj, and Sanjiban Choudhury. Manicast: Collaborative manipulation with cost-aware human forecasting. *ArXiv*, abs/2310.13258, 2023.

[62] Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling open-vocabulary object detection. *arXiv preprint arXiv:2306.09683*, 2023.

[63] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[64] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019.

APPENDIX

We outline the configuration of our system, as detailed in Section A. Subsequently, we delve into further discussions on each component of our system, covered across Sections B, C, and D. Finally, we present an in-depth analysis of the user study, incorporating details on the experimental setup and supplementary findings, all of which are elaborated in Section E.

A. System Setup

Kitchen Scene and Robot Placement. The kitchen scene consists of a main kitchen table at the center where all cooking activities are performed. A pantry is placed near the table, which contains a large range of condiments and kitchen staples. There is also a secondary table on the side of the center table meant for serving up the final dishes. Our robot system includes two robots (R1 and R2).

- R1 (*Franka Emika Research 3* [27]) is a tabletop 7-of manipulator stationed at one end of the kitchen tables at the center of the scene.
- R2 (*Hello Robot Stretch RE1* [26] is a mobile manipulator that can navigate around the kitchen area, capable of fetching and putting away condiments and kitchenware as required by the user.

Camera Placement. For the tabletop manipulator (R1), the perception stack includes two Intel Realsense D435i RGB-D cameras placed above the center kitchen table. Both cameras are placed at opposite ends of the table and at an angle such that they capture the entirety of the tabletop as well as the human user. Integrating both camera perspectives enhances the visibility of objects and human poses within a cluttered kitchen setting, effectively mitigating occlusion issues. The mobile manipulator (R2) uses an onboard Intel Realsense D435i RGB-D head-camera for perceiving objects.

Computational Details. In addition to the onboard computing capabilities of the robots, our setup includes five personal computers (PCs) dedicated to running various system modules. These PCs are connected to the same network, utilizing the Robot Operating System (ROS) for communication. For tasks that demand real-time neural network inference, we employ onboard GPUs, (NVIDIA GeForce RTX 3060). Detailed information about each PC's role and configuration is provided:

- **C1:** Connected with a Bluetooth microphone and speaker, this PC runs the *Speech-Text* system for communicating with the user and the *Interactive Task Planner* that utilizes GPT-4 API calls.
- **C2:** Used for running neural network models related to the perception (object detection) and control (RL agent) of R2. This PC also communicates with C1 to allocate subtasks to R2.
- **C3:** This PC forms the perception stack for R1, including running neural network models for object detection and pose estimation.
- **C4:** This PC runs the human forecasting model using the pose estimates computed by C3. This PC also computes motion plans for R1 based on the predicted object pose and human forecasts. Further, it communicates with C1 to allocate subtasks to the R2.
- **C5:** This PC is installed with a real-time kernel to send joint commands to R2 at 1 kHz frequency as recommended by the robot manufacturers.

B. Interactive Task Planner Details

The interactive task planner consists of three main components: a representation of a recipe and its dependencies, a mechanism to decide on a recipe and assign subtasks to others, and a medium to communicate to robots which skills to use for a given subtask. We implement these using a direct acyclic graph (DAG), a behavior tree, and LLM-generated code (communicated over ROS action services).

Recipe DAG. We represent a recipe as a DAG whose nodes represent subtasks and whose edges represent dependencies. We also maintain a done state for each node to know which subtasks are available next. To determine the available subtasks, we start from a root node whose done state is set, with outgoing edges to the first subtasks. Then, we follow each outgoing edge until reaching a node whose done state is unset and add it to a set. If no node is found through this process, the recipe has been finished. See Figure 12 for an example of a DAG for Caesar salad.

A DAG allows us to represent dependencies, such as, *sequential:* 'do A before B', and *AND* dependencies: 'do A and B before C'; it currently does not allow *OR* dependencies (do A or B before C). However, we could still create 32 unique recipes with this limitation.

Behavior Tree. We use a behavior tree to decide on a recipe and then assign subtasks to others by designing the tree around the behaviors we expect. It takes as input an observation from the world and outputs arguments for high-level actions.

An observation consists of

1) recipe name
2) available subtasks
3) each robot's subtask queue, current subtask, and current status (`Idle`, `Running`, or `Interrupted`)
4) user's subtask queue
5) completed subtask queue
6) user's current input
7) chat history

The recipe name can be empty if the recipe has not been decided yet. The available subtasks are populated by the DAG. The robot and user subtasks are all populated by the behavior tree's high-level actions; the robot additionally has a current subtask and status field updated over a ROS action server as the robots complete their subtasks. When subtasks are completed, the completed subtask queue is updated. Finally, the user's currently spoken input is stored and later appended to the chat history along with the task planner's messages.

The high-level actions include

- `say(msg)`
- `set_recipe(name)`
- `assign(agent, subtasks)`

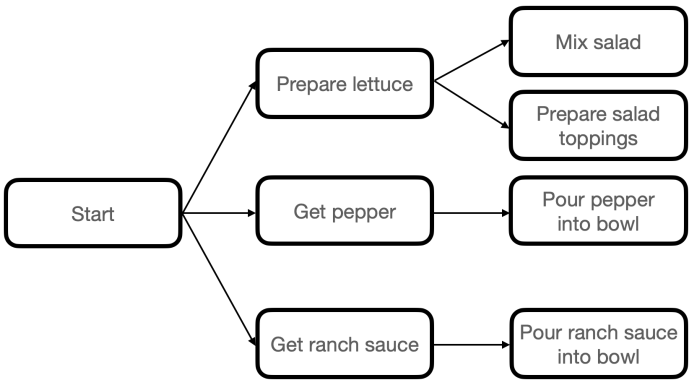


Fig. 12: **Recipe DAG Example.** This DAG represents the subtasks and dependencies involved in making a Caesar Salad. At the beginning of making the recipe, the available subtasks include 'Prepare lettuce', 'Get pepper', and 'Get ranch sauce'. If one of the subtasks is marked as done, the following subtasks become available (e.g. completion of 'Get pepper' causes 'Pour pepper into bowl' to become available).

- `mark_complete(subtasks)`
- `interrupt(agent)`
- `no_op()`

`say(...)` allows the task planner to communicate to the user with a message. `assign(...)` will assign a list of subtasks to an agent (robot or human). `mark_complete(...)` will set a list of subtasks as completed. `interrupt(...)` will stop a robot from doing its current subtask. `no_op()` does nothing.

The tree consists of various nodes that each query an LLM that either outputs (1) a decision for the next node to run or (2) arguments for the high-level actions to take. Each node is associated with a prompt that is used when querying the LLM. If a node's query response is malformed (e.g. bad JSON) or invalid (e.g. bad decision or arguments), the node is rerun. Each node only requires the observation as input, so we can run each node simultaneously to parallelize the LLM queries and draw a path from the root to a leaf based on the decisions made.

The tree runs a cycle to take high-level actions whenever the observation differs from the past observation. This gives the user time to respond to the task planner's questions. To receive user input and respond to the user, we use speech-to-text and text-to-speech modules, respectively. The tree runs indefinitely until the script is terminated.

Code Generation. Whenever the task planner assigns a subtask to a robot, it must be converted into a sequence of low-level skills the robot is capable of. We do this by using an LLM to generate code that the robot runs.

When the task planner assigns a subtask to a robot, it is first added to the robot's subtask queue. A thread dedicated to the robot checks to see if there are any subtasks in the queue, pops it to add to the current subtask, and sets the status to Running. A separate prompt for code generation is used to query an LLM to generate code for the provided subtask. An example of generated code includes

```
from robot_utils import <robot_api>
from env_utils import <env_constants>
pick_up_item(LADLE)
place_item_at(POT)
stir()
```

where `<robot_api>` includes all low-level robot skills like `pick_up_item(...)` and where `<env_constants>` includes enums for objects in the environment. Each line of code executing a robot skill sends a ROS action to the robot to execute said skill. When the robot finishes executing its current skill, it communicates that it has finished to the task planner, which can, in turn, send another skill. This continues until the entire subtask is finished, in which case the robot's current subtask is cleared, and the robot's status is set to Idle. If the robot is interrupted, its current subtask is also cleared, but its status is set to Interrupted.

C. Visuomotor Skills Details

Skill Library The task planner has access to a number of robot skills represented as function calls that are parameterized by object positions and target locations. For each skill, the positions of the objects are estimated using an open-vocabulary object detection model, OWL-ViT (more details in the next section), given text prompts provided by the task planner. For navigation, we store mapped locations to real-world coordinates, assuming the kitchen scene does not change its configuration between runs. We enumerate below the set of low-level skills performed by the two robots in this paper:

1) `pick(<obj>)`: Both robots share the same object detection module to complete the `pick(<obj>)` task to get bounding boxes and a 3D grasp-pose around the object of interest. R1 (Franka arm) moves directly to the grasp pose using an inverse kinematics-based joint impedance controller. R2 (Stretch robot) is tasked with picking up objects from a cluttered pantry. To avoid hitting the pantry and surrounding objects, the robot uses a reinforcement learning policy trained in simulation to execute actions.
2) `move(<loc>)`: This skill uses a map of the kitchen acquired beforehand and the internal localization mechanism of Stretch RE1 to navigate to designated locations around the kitchen.

3) `place(<loc>)`: The `place(<loc>)` skill is parameterized by the target locations and completed with precoded motion primitives.
4) `stir(<obj1, obj2>)` We define this motion primitive for R2 (Franka arm) parameterized by the tool in the robot's hand (`<obj1>`), for example, a ladle and the target utensil where the action takes place (`<obj2>`), for example, a pot. Further, this skill is responsive to the human's movements into the robot's stirring radius. If the human's motion forecasts reach into the robot's workspace, the robot stops stirring and makes space for the human to move in.
5) `pour(<obj1, obj2>)` Similar to the `stir()` function, this skill enables R2 to pour an object (`<obj1>`), such as a salt can into a target receptacle (`<obj2>`), such as a bowl. This process involves the utilization of motion primitives based on the estimated locations of the objects involved. Specifically, in the scenario of pouring salt into a bowl, R2 executes a sequence of actions: it first retrieves the salt can from the table, positions it over the bowl at a calculated tilt angle, and then shakes the can to dispense the salt. Following the completion of the pouring action, R2 returns the salt can to its original location on the table.
6) `handover(<obj>)` R2 (Franka arm) completes handovers quickly and efficiently by directly moving its end-effector towards the forecasted human wrist position. Once the robot's end effector is within a threshold of the human's wrist position, it stops and releases the object into the robot's hand. Finally, the robot arms reset back to its original position.

Object Localization The object localization pipeline first takes as input RGB image and text prompt of the object of interest, which is passed through an OWLViT [62] object detection model that produces k bounding box proposals denoting possible locations of the object. These k bounding boxes are then filtered using non-maximum suppression to remove overlapping boxes. Due to the camera angle and other noise in the environment, we find that the top OWLViT bounding box does not reliably agree with the desired object. Thus, these proposals are refined by feeding each of the images of the cropped bounding boxes and the text prompt to a pre-trained CLIP [23] model to create a CLIP score that measures how aligned each cropped image is with the text prompt[1]. Next, the image, the bounding box with the highest CLIP score, and the text prompt is fed to a pre-trained FastSAM [19] model to segment the object located in the bounding box. The point cloud given by the depth camera is used to project all the points inside the segmentation mask into 3D space. All the 3D points of the object are averaged to obtain a final, single 3D point. This 3D point is then fed to the execution module to produce actions for how the robot should move to the object.

RL as a Differentiable IK Solver The RL agent needs to take the goal prediction and execute a series of actions to safely reach that goal. For `pick()` specifically, consider a pantry that is stocked with items. A desirable trajectory would avoid hitting the pantry boards, hitting neighboring objects, and pushing the object as the gripper approaches. To guide the agent, we create a simulator that, for a given goal point, builds a 3-dimensional set of walls to the sides, back, and bottom of the goal. Invalid actions are those that collide with a wall or violate robot joint states. An episode starts by sampling a start and goal position within some distance reachable by our robot. The observation space is the L_1 norm between the goal and current positions. We then train a Proximal Policy Optimization [63] agent using the implementation from Raffin et al. [64] with the same action space as the teleoperation commands in the demonstration data using the following cost function

$$\exp(-\|O_c - O_g\|_2) - 1 \tag{1}$$

where O_c and O_g represent the current and desired end-effector coordinates respectively, and $\|\cdot\|_2$ is the Euclidean distance. The main failure case for the agent is violating joint constraints while trying to avoid the walls because the observation space does not include joint states.

Behvaioral Cloning Baseline Our BC policy consists of two feed-forward layers with 256 neurons and is trained on 485 demonstration trajectories with variation in the robot base position, robot starting pose, and the goal location on the pantry. At each timestep, the model takes as input the difference between the current end-effector position and the final position (the same as the RL agent), and outputs a 10-dimensional vector of logits, where each dimension corresponds to moving one of the robot's 10 joints. The model is trained using a weighted cross-entropy loss function to account for class imbalances. On-policy, a final action is obtained by categorically sampling from the output vector.

D. Human Motion Forecasting

Model Architecture. We use a Space-Time Separable Graph Convolutional Network (STS-GCN) [60] model architecture for our human-motion forecaster, which encodes the human's joint positions at different timesteps as nodes in a graph. Instead of simply constructing a fully connected graph between all nodes, the model constructs a sparse network without redundant edges across temporal and spatial dimensions. Edges are connected only between the same human joint through consecutive timesteps and between all joints at the same timestep.

Experimental Setup. In order to employ our human motion forecasting model for real-time inference, we make use of an RGB-D camera (Intel RealSense D435) pointed at the human's torso. The human pose is represented by the 3D positions of 7 upper body joints (shoulders, elbows, wrists, and neck). We track the 2D human joint locations using MediaPipe [25] on input RGB images and back-project them to 3D world coordinates using the depth map. As discussed in the approach, our method is forced to handle noisy inputs

[1]If k is set too low, the set may not contain a bounding box around the object of interest to be used by CLIP. If k is set too high, the set of bounding boxes may be too noisy, resulting in lower accuracy. We set $k = 10$ for all experiments.

	Metrics (mm) ↓	BASE	SCRATCH	FINETUNED	FINETUNED-T
REACTIVESTIR	All Joints ADE	60.3 (± 0.6)	40.0 (± 0.3)	32.1 (± 0.2)	29.9 (± 0.2)
	All Joints FDE	91.5 (± 0.9)	60.3 (± 0.5)	54.0 (± 0.5)	51.7 (± 0.4)
	Wrists ADE	83.7 (± 0.6)	58.0 (± 0.4)	47.9 (± 0.3)	44.9 (± 0.3)
	Wrists FDE	128.0 (± 1.0)	87.2 (± 0.7)	80.7 (± 0.6)	76.6 (± 0.6)
	T-All Joints ADE	58.0 (± 0.4)	38.7 (± 0.2)	31.1 (± 0.1)	28.8 (± 0.1)
	T-All Joints FDE	87.7 (± 0.6)	58.0 (± 0.3)	52.0 (± 0.3)	49.6 (± 0.3)
	T-Wrists ADE	81.8 (± 0.4)	56.8 (± 0.2)	46.8 (± 0.2)	43.8 (± 0.2)
	T-Wrists FDE	124.6 (± 0.7)	84.9 (± 0.5)	78.7 (± 0.4)	74.4 (± 0.4)
HANDOVER	All Joints ADE	56.3 (± 0.3)	40.4 (± 0.2)	32.9 (± 0.1)	31.4 (± 0.1)
	All Joints FDE	88.0 (± 0.5)	62.8 (± 0.4)	56.2 (± 0.3)	55.0 (± 0.3)
	Wrists ADE	88.5 (± 0.4)	64.2 (± 0.3)	51.8 (± 0.3)	50.0 (± 0.2)
	Wrists FDE	139.4 (± 0.8)	100.3 (± 0.6)	89.2 (± 0.6)	87.4 (± 0.6)
	T-All Joints ADE	54.0 (± 0.2)	38.9 (± 0.1)	31.7 (± 0.1)	30.2 (± 0.1)
	T-All Joints FDE	83.8 (± 0.4)	59.6 (± 0.3)	53.5 (± 0.3)	52.4 (± 0.3)
	T-Wrists ADE	85.2 (± 0.3)	61.9 (± 0.3)	50.1 (± 0.2)	48.3 (± 0.2)
	T-Wrists FDE	133.0 (± 0.6)	95.4 (± 0.5)	85.2 (± 0.4)	83.4 (± 0.4)
TABLESET	All Joints ADE	107.0 (± 1.1)	72.0 (± 0.5)	59.0 (± 0.4)	59.1 (± 0.4)
	All Joints FDE	181.0 (± 1.9)	118.1 (± 0.9)	108.0 (± 0.8)	108.8 (± 0.8)
	Wrists ADE	127.1 (± 1.0)	93.4 (± 0.6)	80.4 (± 0.5)	81.7 (± 0.5)
	Wrists FDE	224.7 (± 2.0)	152.6 (± 1.1)	143.1 (± 1.0)	145.8 (± 1.0)

TABLE V: **CoMaD Forecasting Metrics.** We report Average Displacement Error (ADE) and Final Displacement Error (FDE) for Handover, Reactive Stirring, and Table Setting tasks on different forecasting models: Base, Scratch, FineTuned, and FineTuned-T. Metrics prefixed with 'T-' indicate measurements from the *transition dataset*, data where humans are in close-contact. Finetuned-T produces the lowest errors on Reactive Stirring and Handover, with very marginally higher errors on Table Setting.

from depth map projections which are out-of-distribution for motion forecasting models trained on high-fidelity motion capture data. We first compare forecasting performance on CoMaD [24] to select a model suitable for predicting human motion in our dynamic kitchen setting. CoMaD is collected via motion capture suits and contains 270 episodes of human-human interactions across 3 different kitchen tasks with an average length of 30 seconds per episode (4+ hours of total data). Then, we conduct experiments injecting various levels of random Gaussian noise into motion capture data at train time to overcome the train-test distribution mismatch and report results on a dataset of human motion tracked by our single-camera setup.

Forecasting Metrics. We quantify errors made by the forecaster by measuring both the Average Displacement Error (ADE) on all predicted timesteps and the Final Displacement Error (FDE) of the predicted pose 1-second prediction into the future given 0.4 seconds of pose history. We report metrics on All Joints as well as Wrists specifically, as they are the most relevant joints in the manipulation tasks we roll out. We additionally report forecasting metrics on the CoMaD *transition dataset* of human motion during short transition windows in which humans come in close contact with one another, denoted by prefix 'T-' (e.g. T-All Joints ADE, T-Wrists ADE). Note that humans are always in very close proximity during the TABLE SETTING task.

CoMaD Forecasting Results. Our two baselines are (1) BASE, trained only on AMASS data, and (2) SCRATCH, trained only on CoMaD data. We report results for two more models: (3) FINETUNED, pre-trained on AMASS data and fine-tuned on CoMaD data, and (4) FINETUNED-T, pre-trained on AMASS data and fine-tuned on CoMaD with upsampling from its *transition dataset*. Each model is tested on a held-out CoMaD test set of episodes. FINETUNED-T significantly outperforms all other models across every metric for the REACTIVE STIRRING and HANDOVER tasks. On the TABLE SETTING task, FINETUNED only marginally produced lower errors compared to FINETUNED-T, both of which beat out the baselines.

We find that upsampling CoMaD transition data where humans are in close contact enables more accurate motion forecasts on kitchen activities. BASE struggles to generate accurate predictions in highly dynamic manipulation tasks, as it was only trained on AMASS [15] data of a single-human and lacks interaction data. SCRATCH is challenged with learning general human motion dynamics from CoMaD, a much smaller

	Metrics (mm) ↓	NOISE 0	NOISE 0.001	NOISE 0.01	NOISE 0.1
REACTSTIR	All Joints ADE	75.1 (± 1.2)	70.8 (± 1.2)	64.8 (± 0.9)	136.2 (± 0.9)
	All Joints FDE	107.3 (± 1.8)	103.5 (± 1.7)	94.0 (± 1.3)	155.4 (± 1.2)
	Wrists ADE	97.6 (± 1.8)	90.4 (± 1.8)	81.8 (± 1.5)	116.0 (± 1.3)
	Wrists FDE	128.1 (± 2.5)	124.5 (± 2.5)	120.7 (± 2.1)	140.3 (± 2.1)
HANDOVER	All Joints ADE	66.1 (± 1.0)	59.9 (± 1.0)	55.2 (± 0.8)	151.1 (± 0.5)
	All Joints FDE	95.9 (± 1.4)	90.6 (± 1.4)	83.2 (± 1.2)	175.6 (± 0.8)
	Wrists ADE	97.5 (± 2.0)	88.0 (± 1.9)	80.1 (± 1.7)	136.0 (± 1.0)
	Wrists FDE	137.8 (± 2.8)	131.0 (± 2.8)	126.8 (± 2.7)	176.8 (± 1.6)

TABLE VI: **Vision-based Forecasting Metrics.** We report Average Displacement Error (ADE) and Final Displacement Error (FDE) for both Handover and Reactive Stirring tasks at various levels of Gaussian noise injection into training inputs ranging from 0 to 0.1. At noise level 0.01, the error is the lowest across all tasks and metrics.

dataset compared to AMASS, which is reflected by its higher errors. Ultimately, we find that pre-training forecasting models on large-scale human activity data and fine-tuning on human-human interaction data yields the best performance in close-proximity kitchen manipulation tasks. Our method employs FINETUNE-T for the remaining experiments.

Vision-Based Forecasting Results. We attempt to address the train-test distribution mismatch (trained on high-fidelity motion capture data and and tested on human poses estimated by RGB-D cameras) faced by the motion forecasting model when making predictions on our RGB-D based 3D pose tracking system by injecting random Gaussian noise to motion capture inputs at train time, forcing the model to denoise inputs and generate smooth forecasts. Formally, we conduct experiments by doing the following: given the history of human pose (J joints, each in 3D coordinates) over the last K timesteps $\phi \in \mathbb{R}^{K\times J\times 3}$, add Gaussian noise $N \in \mathbb{R}^{K\times J\times 3} \sim \mathcal{N}(0, \sigma^2 I)$ to obtain $\phi_\sigma = \phi + N$ (σ denotes the "noise level" injected into the pose history). Let $\xi_H \in \mathbb{R}^{T\times J\times 3}$ denote the human pose in the next T timesteps. Instead of learning a model for $P(\xi_H|\phi)$ as traditional methods do, we learn to model $P(\xi_H|\phi_\sigma)$. Table VI shows vision-based forecasting metrics on the REACTIVE STIRRING and HANDOVER tasks for models trained with $\sigma \in \{0, 0.001, 0.01, 0.1\}$. We find that when forecasting human motion from our single-camera based 3D pose history, the model learned with hyperparameter $\sigma = 0.01$ generates the most accurate predictions across all metrics (ADE and FDE), yielding it most suitable to be integrated into the overall system.

E. Task Planner User Study

Experimental Setup. In order to conduct the user study, we build a web-based application to chat with the task planner. The application is intended to virtually simulate a kitchen environment, where the participants see: 1) the chat history with the planner, 2) the complete recipe, 3) the current task queue of each agent, 4) available tasks, and 5) completed tasks (see figure 13). The application allows users to interact with the task planner once, prepare a pre-determined recipe, and then answer survey questions based on their experience. They are given instructions and examples on how to use the interface, what are each robot's capabilities, and what are the safety constraints to monitor violations of.

We picked 7 recipes: "avocado toast", "sundae", "milk-shake", "biryani", "ramen", "stir fried noodles" and "pasta", to assign to participants in the internal study, randomly selecting a mixture of desserts, noodles, and entrées with roughly the same number of nodes in their recipe DAG. Each participant prepared the same recipe twice, one with each planner (*One-Prompt* and *Tree*), but was not made aware that the planner was different in the two interactions.

We also picked 10 recipes: "mango sticky rice", "eggdrop soup", "pasta salad" and 6 from above, to conduct the external study. We again added a variety of different recipes of similar length. We notably excluded "biryani", as our internal study showed participants from all regions and cultures may not be familiar with this dish, and familiarity of a recipe helps them focus on the interaction.

Therefore, out of the $n = 46$ interactions, 26 were from 13 internal participants, set up as a within-study, and 20 were from unique external participants, set up as an in-between study. We randomize the order in which the two planners are presented in the internal study, and all participants participated in a "pilot" study with a recipe of their choice to develop familiarity with the interface. We do not include results from the "pilot" study due to a large variance in interactions depending on the size of the recipe.

Instructions to the Users. We asked all users - both internal and external, to interact with the planner naturally and with at least 3 non-nominal interactions (in order to bring out constraint violations)

1) They should not directly name the recipe they want to make, and instead lead the assistant into suggesting it.
2) They must make at least one intervention in the assignment of a task, like "I will handle the stirring"
3) They must add at least one task that was not part of the

Act Without Permission	*Observation*	The recipe is Pasta Salad. The available subtasks are 'get salt', 'get pepper', 'mix bowl', and 'prepare vegetables'. The latest user input asks if salt and pepper are the next ingredients.
	Action	`say("Yes, salt and pepper are next. I'll send R2 to get them for you.")` `assign("R2", ['get salt', 'get pepper'])`
	Error Analysis	The One-Prompt assigns 'get salt' and 'get pepper' to R2 before user confirmation. This is against the constraint that the One-Prompt cannot directly assign without confirming.
Lying	*Observation*	The recipe is Corn Soup. There are no available subtasks. The latest user input states that they want to pour corn into the pot instead of R1.
	Action	`say("Understood, you will handle pouring the corn into the pot. Please let me know if there's anything else I can assist with!")`
	Error Analysis	One-Prompt acknowledges the user and confirms that they will handle pouring the corn into the pot but does not assign this to the user. This is a lie because the user was never assigned the task despite what the task planner claims.
Ignore User	*Observation*	The recipe is Pasta Salad. The available subtasks are 'mix bowl', 'prepare vegetables', 'pour salt into bowl', 'pour pepper into bowl', and 'pour pasta into bowl'. The latest user input asks R1 to prepare the vegetables and says that the user will mix the bowl.
	Action	`say("Understood. You will prepare the vegetables and mix the bowl. Please let me know once you have finished preparing the vegetables, and I can assist with the next steps for the pasta salad. R1 will remain on standby for now")` `assign("User", ['prepare vegetables', 'mix bowl'])`
	Error Analysis	One-Prompt ignores the user's request to assign 'prepare vegetables' to R1. Though this is outside of R1's capabilities, a message of 'Unfortunately, this is not within R1's capabilities. Would you like to prepare the vegetables?' would have acknowledged the user's instructions without ignoring them. By ignoring the user's request for R1, One-Prompt is against the constraint to reply to all of the user's instructions.

TABLE VII: Examples of constraint violations using the *One-Prompt* planner. *Observation* describes the situation; *Action* lays out the action taken by the task planner; *Error Analysis* explains why this is a violation and what is the correct response/action.

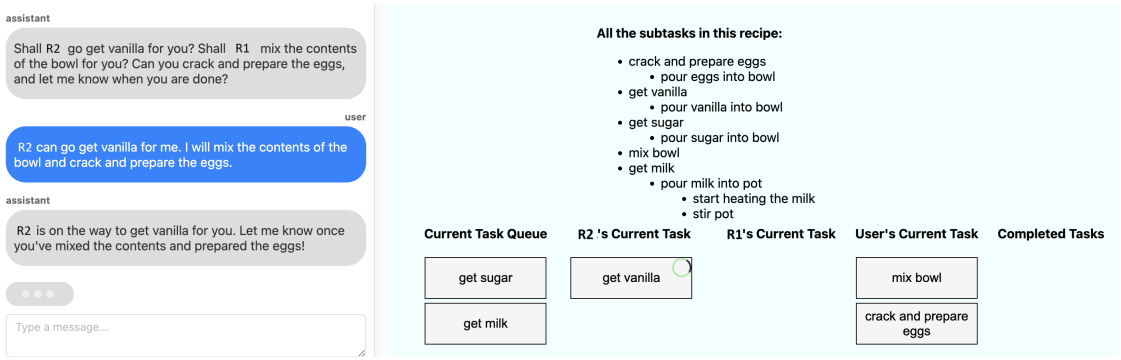


Fig. 13: Chat Page simulating interaction with the task planner for the user study. Includes chat window (left), list of subtasks in recipe (top right) and queues of current, assigned and completed subtasks (bottom right)

recipe, like "get me eggs" for Ramen.

As part of the post-chat survey, we ask the users the following questions:

1) How many times has the assistant assigned a task without their permission?
2) How many times they were lied to by the assistant?
3) How many times did they feel ignored by the assistant?

Their chat history is presented to them as they fill out this survey, and they are asked to provide specific instances along with each answer. Three authors then cross-validated the users' answers with the chat history

Full Quantitative Results Table VIII shows the results of our study on the three metrics we discussed above. We see that while each study by itself shows some trends, both studies put together give us enough data to reject the null hypothesis along two metrics (lying and assigning without confirmation). We also see that the overall frequency of ignoring the user is low in both approaches.

Result Analysis We provide examples for how *One-Prompt* and *Tree* violate each constraint:

Study	Approach	Act Without Permission		Lying		Ignore User	
		M ± SE	t, p, df	M ± SE	t, p, df	M ± SE	t, p, df
Combined Study (n = 46)	**One-Prompt** **Tree**	2.26 ± 0.42 1.22 ± 0.26	$-2.1, \mathbf{.04}, 36.5$	1.39 ± 0.31 0.56 ± 0.24	$-2.11, \mathbf{.04}, 41.76$	0.35 ± 0.15 0.30 ± 0.15	$-0.21, .83, 43.9$
Internal Study (n = 26)	One-Prompt Tree	2.15 ± 0.42 1.53 ± 0.38	$-1.07, .29, 24$	1.23 ± 0.32 0.3 ± 0.17	$-2.51, .02, 24$	0.23 ± 0.12 0.53 ± 0.24	$1.13, .27, 24$
External Study (n = 20)	One-Prompt Tree	2.4 ± 0.83 0.8 ± 0.29	$-1.8, .08, 24$	1.6 ± 0.58 0.90 ± 0.50	$-0.91, .37, 18$	0.50 ± 0.30 0.00 ± 0.00	$-1.63, .12, 18$

TABLE VIII: Results from the User Study(s), which show significant reduction in *Act Without Permission* and *Lying* (n = 46) with Tree Task Planner. M: *Mean*, SE: *Standard Error*, t: *t-value*, p: *p-value*, df: *degrees of freedom*

- *Act Without Permission*: The task planner assigns/removes subtasks without user's permissions.
- *Lying*: The task planner claims to do something but does not do it.
- *Ignore User*: It does not respond to the user's instruction.

Table VII lists examples of violations for each of these constraints.