From Vague Instructions to Task Plans:
A Feedback-Driven HRC Task Planning Framework based on LLMs

Afagh Mehri Shervedani[1], Matthew R. Walter[2], and Miloš Žefran[1]

Abstract— Recent advances in large language models (LLMs) have demonstrated their potential as planners in human-robot collaboration (HRC) scenarios, offering a promising alternative to traditional planning methods. LLMs, which can generate structured plans by reasoning over natural language inputs, have the ability to generalize across diverse tasks and adapt to human instructions. This paper investigates the potential of LLMs to facilitate planning in the context of human-robot collaborative tasks, with a focus on their ability to reason from high-level, vague human inputs, and fine-tune plans based on real-time feedback. We propose a novel hybrid framework that combines LLMs with human feedback to create dynamic, context-aware task plans. Our work also highlights how a single, concise prompt can be used for a wide range of tasks and environments, overcoming the limitations of long, detailed structured prompts typically used in prior studies. By integrating user preferences into the planning loop, we ensure that the generated plans are not only effective but aligned with human intentions.

I. INTRODUCTION

Planning is a fundamental aspect of robotics, enabling autonomous agents to generate sequences of actions to achieve specific goals. Traditional planning methods in the context of human-robot collaboration (HRC) and assistive robots can be broadly categorized into two main types: **rule-based planners** and **learning-based planners**. Rule-based planners rely on predefined heuristics and symbolic representations, making them interpretable at the expense of not being able to adapt to complex or dynamic environments. In contrast, learning-based planners, particularly those utilizing deep reinforcement learning, learn to generate plans from experience in an adaptive manner. However, they often require extensive amounts of training data and suffer from issues such as poor generalization and a lack of interpretability.

The primary objective of **task-level planning** is to decompose complex tasks into manageable subtasks and sequence them effectively to achieve the ultimate goal. Traditional approaches to task-level planning are **rule-based** and often employ a hierarchical model of the dependencies between actions using Hierarchical Task Networks (HTNs) [1, 2]. Over time, various methods have been introduced to enhance the expressive power and adaptability of HTNs. For example, the Planning Domain Definition Language (PDDL) was integrated into HTNs to improve their ability to handle
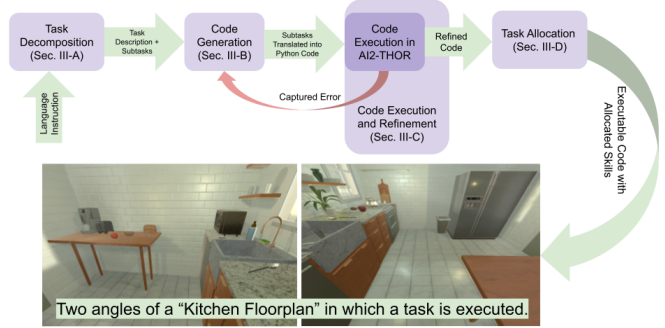

Fig. 1: Overview of the **IteraPlan** framework. The framework consists of four key components: (1) Task Decomposition, where vague, high-level instructions are transformed into structured task descriptions and subtasks; (2) Code Generation, where the subtasks are translated into executable Python code based on available skills and environmental constraints; (3) Real-Time Execution and Adaptive Code Refinement, where the generated code is executed within the AI2-THOR simulation environment, errors are detected, and iterative refinements are made to ensure feasibility; and (4) Affordance-Based Task Allocation, where executable code is adjusted to allocate actions between human and robot agents based on their capabilities.

complex planning problems [3]. Work by Nooraei et al. [4] and Mohseni-Kabir et al. [5] explores how robots can learn hierarchical task models from demonstrations by integrating learning from demonstration (LfD) with Hierarchical Task Networks (HTNs), allowing them to construct HTNs dynamically rather than relying solely on predefined structures.

Building on progress in machine learning, researchers deployed probabilistic decision-theoretic methods such as **Markov decision processes (MDPs)** [6] and **reinforcement learning (RL)** that replace rule-based planners with data-driven task-level planning methods for HRC. Unhelkar and Shah [7] model collaborative human-robot tasks using partially observable Markov decision processes (POMDPs), specifically focusing on a human-robot handover task in a kitchen setting where a robot assists a person in preparing sandwiches. The task involves multiple steps, including fetching ingredients from cabinets, assembling them in cooking areas, and performing handovers. Huang et al. [8] introduce a POMDP-based adaptive task planning model that accounts for both designer and operator intents in human-robot assembly. The study presented in Rusu et al. [9] explores how RL can be used to train assistive robots to autonomously break down complex tasks (e.g., helping humans dress, fetch items, etc.) into subtasks, plan actions, and adapt to varying human needs.

However, while an improvement over rule-based methods, the aforementioned learning-based approaches require exten-

sive domain-specific engineering (e.g., in defining the set of states and actions) and as a result, exhibit limited generalizability to new environments and tasks. This has led to increasing interest in more adaptive and data-driven planning approaches that take advantage of the large-scale reasoning capabilities of large language models (LLMs). Among them, SayCan [10] employs an LLM (PaLM [11]) to decompose a task into a sequence of candidate subtasks that they then rank in terms of their feasibility with respect to the environment and robot capabilities using an affordance model. The model conditions its plans on the basis of both language input and real-world feasibility, enabling robots to perform assistive tasks, such as fetching objects or cleaning. Additionally, SMART-LLM [12] uses LLMs for task-level planning in a multi-agent environment. SMART-LLM converts high-level task instructions provided as input into a multi-robot task planning problem, allowing agents to coordinate and execute tasks with high-level guidance.

Despite these advances, there remains a gap in task-level planning that seamlessly integrates human feedback, adaptive learning, and real-time interaction, particularly in complex, dynamic, and collaborative environments. Furthermore, most LLM-powered systems focus on simplifying planning through textual input without addressing the underlying nuances of human preferences and dynamic task modifications.

Contributions of IteraPlan: In this paper, we address the challenges in task-level planning for HRC by proposing **IteraPlan** (Fig. 1), an adaptive framework that leverages the strengths of LLMs, while incorporating continuous human feedback to dynamically adjust task plans. Specifically, the paper makes the following contributions:

- Unlike existing studies, where LLMs decompose *explicit* high-level task instructions into subtasks, we address a more complex scenario in which the task description is *implicit* in the high-level human's natural language command, such as a user stating, "I feel hungry, what do you suggest I do?" This requires deeper reasoning and task extraction from vague or incomplete user instructions.
- We use a single, concise prompt for task decomposition across all types of tasks and scenes, in contrast to other LLM-based approaches that rely on long, detailed structured prompts tailored to specific tasks. This generalizability allows our framework to adapt seamlessly to a wide range of tasks and unforeseen circumstances, without requiring extensive prompt engineering.
- Unlike other studies where the LLM-generated plan is a fixed and non-refined output we propose a feedback-driven mechanism that fine-tunes the plan iteratively by incorporating feedback from the environment. The system adjusts the plan based on real-world interaction and accounts for user preferences.
- We demonstrate the effectiveness of our framework on a series of human-robot collaboration tasks, showing how it can be applied in practical assistive settings.
- We open-source the code and implementation of this

study in: https://github.com/roboticslabuic/IteraPlan.git

II. RELATED WORK

A. Rule- and Learning-based Task-Level Planning

1) Rule-based Planning: Rule-based planners rely on predefined heuristics and symbolic representations that can be more interpretable but often fail to generalize to dynamic environments. One of the earliest and widely used rule-based interaction managers was Kismet [13], which responded to social cues using predefined rules. Similarly, the Robovie robot series [14, 15] leveraged rule-based architectures to manage interactions, utilizing episode chains to dynamically adapt behaviors based on human responses. While effective for structured interactions, these approaches were rigid and struggled with off-script scenarios.

Hierarchical approaches, particularly HTNs [1, 2], were introduced to improve scalability by decomposing tasks into sequences of subtasks. SHOP [16] and its successor SHOP2 [3] expanded the capabilities of HTNs by incorporating the PDDL, allowing for more expressive task representations. However, despite these advances, rule-based hierarchical planners remained limited in their ability to generalize to new tasks and to dynamically adapt. Research efforts have attempted to mitigate these issues by integrating LfD into HTNs [4, 5], where human guidance aids robots in structuring task sequences. Extensions such as Task Insertion HTNs (TIHTNs) [17] introduced additional flexibility, yet rule-based systems continued to struggle with adaptability and real-time learning.

2) Learning-based Planning: To overcome the limitations of rule-based planners, researchers have turned to probabilistic decision-theoretic approaches such as MDPs [6] and RL. MDPs have been particularly useful for task-level planning under uncertainty, as demonstrated by Gopalan and Tellex [18], where a POMDP model enabled robots to infer human intentions based upon multimodal cues. The survey by Kurniawati [19] further highlighted the computational challenges of POMDPs, but pointed to advances in sampling-based solvers as a means to improve their practical applicability.

Reinforcement learning has gained traction due to its ability to learn optimal policies through interaction. Deep RL methods [20, 21] have demonstrated significant success in robotics, with applications extending to assistive tasks [9]. Inverse RL techniques, such as preference-based learning [22], have been used to infer user preferences and to personalize interactions. However, many RL-based planners require a significant number of environment interactions and extensive training data, lack interpretability, and struggle with long-horizon task decomposition. While these methods improve adaptability compared to rule-based approaches, most existing methods do not provide a means of incorporating online human feedback in real-time, which is crucial for human-robot collaboration (HRC).

B. Large Language Models (LLMs) for Planning

Recently, LLMs have emerged as a promising tool if not an alternative for planning, due to their ability to process

high-level task descriptions and generate flexible action sequences. SayCan [10] integrates an LLM (PaLM [11]) with a robotic affordance model, enabling task decomposition and execution based on real-world feasibility. Similarly, SMART-LLM [12] employs LLMs for multi-agent task planning, allowing coordination through natural language inputs. While these approaches demonstrate the utility of LLMs for planning, they often rely on structured prompts and are unable to dynamically adjust plans based on real-time human feedback.

In the domain of dialogue systems, Generative Pre-trained Transformers (GPT)-based planners have shown potential in sequential decision-making tasks. GPT-Critic [23] utilizes offline RL to fine-tune LLMs for dialogue planning, improving response generation through iterative feedback. LSTDial [24] extends this idea by incorporating short- and long-term feedback mechanisms. While these methods enhance adaptability in textual planning, their applicability to multimodal human-robot interaction remains limited. Unlike traditional planners, LLMs struggle with ambiguity in user input and require mechanisms to refine their outputs based on continuous interaction.

C. Bridging the Gap: Adaptive LLM-based Planning with Human Feedback

Despite advances in rule-based, learning-based, and LLM-based planning, existing approaches face challenges in generalization, interpretability, and adaptability. Rule-based systems are rigid, learning-based planners require extensive data, and LLMs often generate plans in a one-shot manner without real-time refinement. OUr proposed approach addresses these limitations by leveraging LLMs for task-level planning while integrating continuous human feedback. Unlike existing LLM-based planners, our method dynamically refines plans based on environmental interactions, enabling adaptive execution in complex, collaborative scenarios. The proposed system addresses challenges such as implicit task extraction, real-time user preference integration, and iterative plan refinement, ultimately enhancing human-robot collaboration in assistive settings.

III. METHODOLOGY

In this section, we describe our approach to task-level planning for HRC using LLMs. **IteraPlan**, as shown in Fig. 1, consists of four main components: (1) Task Decomposition from Vague Instructions, (2) Translation of Task Plans into Executable Code, (3) Real-Time Execution and Adaptive Code Refinement, and (4) Affordance-Based Task Allocation.

A. Task Decomposition from Vague Instructions

The first component of our approach involves transforming vague, high-level instructions into actionable task plans. Unlike previous methods, which typically start with well-defined, feasible high-level instructions and detailed environmental information, we aim to work with more ambiguous inputs. This introduces two primary challenges: first, ensuring that the task description generated based on a

vague instruction is feasible in the given environment; and second, preventing the inclusion of unnecessary or redundant information about the environment in the planning process.

To address these challenges, we break the task decomposition process into two stages:

a) Stage 1: Initial Task Description Generation: In the first stage, we provide the LLM with the vague instruction and a basic list of relevant locations and objects in the environment. The LLM uses this information to generate an initial **Task Description**—a high-level outline of the task—and a corresponding set of **subtasks** that detail the steps required to complete the task. At this stage, the model may generate a task description that is not fully compatible with the environment due to the potential vagueness of the instruction that lacks specific guidance regarding the feasibility of the plan.

b) Stage 2: Refinement and Feasibility Check: To ensure that the generated plan is both feasible and relevant, the second stage involves refining the prompt. Using the initial Task Description and its associated subtasks from Stage 1, we update the prompt to include only the relevant environmental details: specifically, the objects and locations directly involved in the execution of the subtasks. This ensures that unnecessary details are excluded, minimizing the risk of confusion and the inclusion of irrelevant information. The LLM then refines the Task Description and subtasks based on the updated prompt, aligning the plan with the constraints of the environment and ensuring the feasibility of the task.

By decomposing the task decomposition into these two stages, we ensure that the LLM generates a practical and context-aware task plan while focusing on the elements of the environment that are essential to the task. This approach allows for effective handling of vague instructions without overwhelming the planning process with excessive information, an approach that has proven effective in other domains [25].

B. Translation of Task Plans into Executable Code

The second component of our methodology focuses on translating the task plan and subtasks generated in the previous component into executable Python code. This step is crucial in bridging the gap between high-level natural language planning and real-world execution within the experimental environment.

To achieve this, the LLM receives:
- The refined **Task Description** and **subtasks** from the previous component.
- A structured prompt that contains the set of **available skills** (the term *Skill* will be explained in Section IV-C) within the experiment environment.
- The **properties of objects and locations** involved in the task, including constraints on feasible actions.

Using this information, the LLM generates a Python script that implements the planned sequence of *skills*. However, since automatic code generation may not always result in

an immediately executable or optimal script, we incorporate **human feedback** into this process.

After receiving the initially generated Python code, a human collaborator evaluates its correctness, feasibility, and alignment with task objectives. The code is presented in a terminal window, where the user can review it before execution. Since the generated code consists of a sequence of predefined skills, the user ensures that all referenced skills exist in the simulation environment and that objects used in the script are present in the scene. Additionally, because the system operates with vague high-level instructions, plan correctness is often subjective. The user can inject personal preferences, modifying the plan to better align with their needs.

To facilitate user interaction, the system prints the generated code in the terminal, followed by a prompt asking whether the script appears correct. If the user identifies missing elements, or incorrect assumptions, or wishes to customize the plan (e.g., modifying task details such as ingredient choices in a cooking scenario), the system requests specific feedback and iteratively refines the code based on user input. The process of fine-tuning continues until the human confirms that the generated code meets all requirements.

Importantly, **no programming expertise** is required. The generated code consists of intuitive, human-readable skill sequences, where each skill is associated with specific objects. For example, the command `SliceObject('Apple', 'Knife')` clearly indicates that the slicing skill is applied to an apple using a knife. This structured representation ensures that users can easily understand and modify the code without needing prior knowledge of Python or programming.

By integrating human feedback into the code generation process, this component ensures that the final execution aligns with both environmental constraints and user-specific preferences, making the system more adaptable and reliable in real-world human-robot collaboration scenarios.

C. Real-Time Execution and Adaptive Code Refinement

The third component of our methodology focuses on executing the generated code in the simulation environment using an agent. This agent is assumed to be **a perfect agent**, meaning it can perform all available skills within the environment flawlessly. However, unforeseen circumstances, such as environmental physics or unmodeled constraints, may still cause execution failures. To ensure robust execution under such conditions, we introduce a mechanism for **real-time task execution and adaptive code refinement**.

The initially generated Python code is executed in the simulation environment, and any failures that occur due to unforeseen conditions are captured. These failures could arise due to:

- Environmental constraints that were not explicitly encoded in the task description.
- Physical interactions or dependencies that affect the feasibility of an action.

Whenever execution fails (e.g., due to collision constraints, or invalid skill execution), the system extracts error messages directly from the simulation environment's execution logs. These messages, which may include exception traces or structured failure reports, are parsed and converted into a prompt. This prompt is then provided to the LLM as feedback, guiding it to refine the task plan iteratively. The LLM adjusts the code by incorporating new constraints, modifying execution sequences, or selecting alternative skills to ensure successful task completion, an interactive approach that has proven effective as a means to improve the reasoning capabilities of LLMs [25, 26]..

The LLM refines the code by addressing the specific failure while ensuring that subsequent steps in the task plan remain valid. This refinement process is iterative: each modified version of the code is re-executed, errors are re-evaluated, and further refinements are made until the code executes successfully. This dynamic adaptation ensures robust task execution without requiring predefined, rigid plans.

By integrating real-time execution monitoring with iterative refinement, this component ensures that the framework remains robust against unpredictable environmental conditions. The ability to dynamically correct and adjust execution plans enhances the overall reliability of task completion in human-robot collaboration scenarios.

D. Affordance-Based Task Allocation

The final component of our methodology focuses on the collaborative execution of tasks by both the human and the robot agents. To achieve seamless task distribution, we introduce an **affordance-based task allocation** mechanism that assigns skills based on the capabilities of each collaborator.

Each *skill* is a sequence of *atomic actions*, where each atomic action is associated with an affordance value that indicates which agent can perform the action based on their capabilities. The Task Allocation Component assigns skills to agents using these affordances as follows:

- Extract the skills involved in the plan from the Python code generated in the previous stage.
- Compute affordance values for each skill based on the affordances of its atomic actions using rule-based heuristics.
- Generate a prompt containing the skills and their affordances and employ an LLM solely to revise the Python code, ensuring that each skill is executed by the agent best suited to the skill according to their affordance. The LLM does not reason about affordances, but instead only modifies the syntax to match the assignment.

The system ensures: 1) Each skill is assigned to the agent (human or robot) best suited for efficient execution. 2) Task sequencing is preserved, maintaining a coherent and goal-driven collaboration. 3) Execution remains smooth, with both agents contributing in a synchronized manner.

The affordance-based task allocation strategy enables **adaptive role distribution**, allowing the framework to dynamically adjust assignments based on environmental constraints, agent availability, or task complexity. These factors are incorporated into the heuristics used for affordance calculations, ensuring flexibility in execution.

IV. IMPLEMENTATION

In this section, we describe the technical aspects of our framework, including the LLM that we used, the experimental environment, and the set of available actions.

A. Large Language Model (LLM)

Our framework is compatible with different versions of GPT, such as GPT-4o, GPT-4, and GPT-3.5, ensuring broad applicability across different LLM architectures. The model interacts with the human collaborator for iterative feedback and fine-tuning to improve task execution.

B. Experimental Environment

The experiment was conducted in the AI2-THOR simulation environment [27], a photorealistic interactive framework for embodied AI research. AI2-THOR provides a diverse set of household environments where agents can interact with objects, navigate spaces, and perform various manipulation tasks. To align AI2-THOR's action space with real-world execution, modifications were made to the available set of actions, ensuring that the generated plans could be executed both in simulation and in real-world scenarios.

C. Action Space and Skill Definition

The action space within the AI2-THOR environment was modified to enhance compatibility with real-world execution. Actions were selected based on their feasibility in both simulated and physical settings. Each object and location in the environment has specific affordances that define the possible interactions an agent (robot or human) can perform. To structure these interactions, we define a set of atomic actions as fundamental operations that an agent can execute. These atomic actions serve as building blocks for higher-level skills, which are structured sequences of actions enabling the execution of more complex, goal-oriented behaviors.

We formally define the set of atomic actions and their composition into structured skills. The available actions in the environment are presented in Table I. Each of those actions represents a fundamental capability that can be combined into higher-level skills. Skills are defined as sequences of atomic actions, enabling structured execution of complex tasks. Table II gives a summary of defined skills.

As mentioned above, we modify the set of atomic actions in the simulation environment as described in Table I and introduce the set of skills as defined in Table II since we want our framework to be compatible with real-world executions and be applied in practical assistive settings. For example, the action **SliceObject** in the AI2-THOR environment is contextual and doesn't require any *Tool Object*. However, we enforce using a sharp tool object for performing this action. The same holds for action **CleanObject** with enforcing usage of a *Tool Object* (e.g., Towel, Scrub Brush) and a *Detergent Object* (e.g., Soap, Spray Bottle).

Action	Description
GoToObject ⟨objectId⟩	Navigates towards an object.
OpenObject ⟨objectId⟩	Opens an object.
CloseObject ⟨objectId⟩	Closes an object.
BreakObject ⟨objectId⟩	Breaks an object.
SliceObject ⟨objectId⟩, ⟨toolObjectId⟩	Slices an object using a tool.
SwitchOn ⟨objectId⟩	Switches on an object.
SwitchOff ⟨objectId⟩	Switches off an object.
CleanObject ⟨objectId⟩, ⟨toolObjectId⟩, ⟨canBeUsedUpDetergentId⟩	Cleans an object using a tool and detergent.
PickUpObject ⟨objectId⟩	Picks up an object.
PutObject ⟨receptacleObjectId⟩	Places picked-up object inside a receptacle.
ThrowObject ⟨objectId⟩	Throws an object.
UseUpObject ⟨objectId⟩	Uses parts of an object (such as Tissue or Soap).

TABLE I: Set of atomic actions available in the AI2-THOR environment.

Skill	Sequence of Actions
Open an Object	GoToObject ⟨objectId⟩ → OpenObject ⟨objectId⟩
Close an Object	GoToObject ⟨objectId⟩ → CloseObject ⟨objectId⟩
Break an Object	GoToObject ⟨objectId⟩ → BreakObject ⟨objectId⟩
Clean an Object	GoToObject ⟨toolObjectId⟩ → PickupObject ⟨toolObjectId⟩ → GoToObject ⟨canBeUsedUpDetergentId⟩ → UseUpObject ⟨canBeUsedUpDetergentId⟩ → GoToObject ⟨ObjectId⟩ → CleanObject ⟨ObjectId⟩
Switch On an Object	GoToObject ⟨ObjectId⟩ → SwitchOn ⟨ObjectId⟩
Switch Off an Object	GoToObject ⟨ObjectId⟩ → SwitchOff ⟨ObjectId⟩
Put an Object in a Receptacle	GoToObject ⟨ObjectId⟩ → PickupObject ⟨ObjectId⟩ → GoToObject ⟨receptacleObjectId⟩ → PutObject ⟨receptacleObjectId⟩
Slice an Object	GoToObject ⟨toolObjectId⟩ → PickupObject ⟨toolObjectId⟩ → GoToObject ⟨ObjectId⟩ → SliceObject ⟨ObjectId⟩ → ThrowObject ⟨toolObjectId⟩

TABLE II: Set of skills defined as sequences of atomic actions.

V. EXPERIMENTAL EVALUATIONS

A. Preliminary Study

To evaluate the performance of our framework, we conducted experiments in the AI2-THOR simulation environment across four different floorplan types: **Kitchen**, **Living Room**, **Bedroom**, and **Bathroom**. We performed a total of 40 tests. In **Kitchen**, we considered four different tasks (Task 1: "I feel hungry." Task 2: "Can you make me sunny side up egg?" Task 3: "I feel so sleepy and tired." Task 4: "Can you help me empty the cabinets and drawers?") across five distinct kitchen floorplans, resulting in a total of 20 tests. In the **Living Room**, we considered a single task (Task 1: "Can you help me tidy up the room?") across 5 different living room floorplans, totaling five tests. In the **Bedroom** floorplan, we looked at a single task (Task 1: "I feel so tired, need to relax.") across five different bedroom floorplans, totaling five tests. For the **Bathroom** floorplan, we examied two different tasks (Task 1: "I feel so tired, need to relax." Task 2: "Can you help with cleaning the room?") across five different bathroom floorplans, leading to a total of ten tests. Each of these tests was repeated using three different LLMs: GPT-4o, GPT-4, and GPT-3.5-turbo. This resulted in a total of 120 tests conducted across all floorplan types and models.

Performance was evaluated based on floorplan-wise task planning. As described in previous sections, a single concise prompt was used for task planning across different floorplan types. Unlike other state-of-the-art approaches that utilize structured prompts with detailed descriptions and numerous examples spanning multiple floorplans, our approach relies on a single example in the prompt, expecting the LLM to generalize across all floorplans. Therefore, our evaluation focuses on how well each model adapts and performs across different floorplan types without explicit training on extensive examples.

We focused on evaluating **Component 2 (III-B)** and **Component 3 (III-C)** of the framework, as **Component 1** and **Component 4** consistently perform flawlessly and do not require further assessment. Since the core planning processes occur in Components 2 and 3, our evaluation metrics are designed to measure their effectiveness.

For **Component 2**, which involves generating the optimal executable code, we defined a constraint on the maximum number of refinements allowed, ranging from 0 to 5. We measured the **Optimal Code Rate (OCR)**—the success rate of obtaining the optimal code if 0, 1, 2, 3, 4, or 5 interactions (i.e., instances of human feedback and fine-tuning) were allowed. Table III shows the K-OCR, L-OCR, Bd-OCR, and Bt-OCR values for Kitchen, Living Room, Bedroom, and Bathroom floorplans respectively. Comparing the last two rows of the table, we see that allowing even only one refinement improves the performance of Component 2 significantly. GPT-4o achieves optimal executable code faster than other models, reaching OCR of 1.00 with fewer refinements in Living Room, Bedroom, and Bathroom tasks (at 2 refinements). GPT-4 needs slightly more refinements to reach 1.00 in some cases, such as the Bedroom. GPT-3.5 lags behind, requiring four refinements in the Kitchen before achieving a 1.00 OCR, indicating its difficulty in generalization. GPT-4 performs best in the Living Room task, reaching 1.00 at just 1 refinement, while GPT-4o struggles more, requiring 2 refinements.

For **Component 3**, which involves executing tasks successfully, we again defined a constraint on the maximum number of refinements allowed, ranging from 0 to 5. Since our framework employs adaptive code refinement and real-time task execution, all tasks eventually succeed. However, we analyzed how many interactions (i.e., feedback and fine-tuning cycles) were necessary to reach this guaranteed success. We measured the **Successful Execution Rate (SER)**—the rate of successful executions if 0, 1, 2, 3, 4, or 5 interactions were allowed. Table IV shows the K-SER, L-SER, Bd-SER, Bt-SER values for Kitchen, Living Room, Bedroom, and Bathroom floorplans respectively. In most cases, GPT-4 reaches a 1.00 SER slightly earlier than GPT-4o, particularly in the Living Room (at 2 refinements). All models require more refinements (GPT-3.5, GPT-4, and GPT-4o reaching 1.00 SER at 4, 3, 5 refinements respectively) in the Kitchen. This is due to tasks being more complex in the Kitchen. Notably, the zero-shot performance (SER at 0 refinements) varies significantly across different GPT models. Overall, the table highlights the ability of LLM to adequately revise the plan if the execution fails due to unforeseen conditions of the environment.

	K-OCR			L-OCR			Bd-OCR			Bt-OCR		
	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5
5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	**1.00**	**1.00**	0.95	1.00	1.00	1.00	1.00	**1.00**	1.00	1.00	1.00	1.00
2	0.95	0.95	0.95	**1.00**	1.00	1.00	**1.00**	0.80	1.00	**1.00**	1.00	1.00
1	0.85	0.95	0.75	0.60	**1.00**	**1.00**	0.60	0.80	**1.00**	0.80	**1.00**	0.60
0	0.30	0.45	0.60	0.00	0.20	0.00	0.60	0.60	0.20	0.40	0.20	0.00

TABLE III: OCR Results for Different Floorplan Types

	K-SER			L-SER			Bd-SER			Bt-SER		
	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5
5	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
4	0.95	1.00	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	0.95	**1.00**	0.95	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.95	0.95	0.95	0.80	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	0.95	0.90	0.95	0.60	0.80	**1.00**	**1.00**	1.00	**1.00**	1.00	**1.00**	**1.00**
0	0.70	0.75	0.85	0.40	0.60	0.40	0.60	**1.00**	0.40	1.00	0.80	0.90

TABLE IV: SER Results for Different Floorplan Types

B. Stochasticity Study

Due to non-determinism in LLMs behavior, they produce non-deterministic responses particularly when it comes to code generation [28]. To assess this, we calculate and report mean and standard deviation values for OCR and SER across different floorplans. The results summarized in Table V demonstrate that when no refinements are allowed or only one refinement is allowed, the results are slightly inconsistent. That is due to the fact that we used a single concise prompt for task planning across different floorplan types. Nevertheless, allowing 2 or more refinements makes LLM responses more consistent.

	OCR			SER		
	GPT-4o	GPT-4	GPT-3.5	GPT-4o	GPT-4	GPT-3.5
5	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
4	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
3	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.02	0.98 ± 0.02	1.00 ± 0.00	0.98 ± 0.02
2	0.98 ± 0.02	0.93 ± 0.09	0.98 ± 0.02	0.93 ± 0.09	0.98 ± 0.02	0.98 ± 0.02
1	0.71 ± 0.13	0.93 ± 0.09	0.83 ± 0.19	0.88 ± 0.19	0.92 ± 0.09	0.98 ± 0.02
0	0.32 ± 0.25	0.35 ± 0.19	0.20 ± 0.28	0.67 ± 0.25	0.78 ± 0.16	0.63 ± 0.27

TABLE V: OCR and SER Mean $\pm$ STD Across All Floorplans

C. Ablation Study

As we mentioned, **Component 4 (III-D)** of the framework performs flawlessly since it is rule-based. However, we are interested in evaluating its performance under a different approach: instead of using heuristics and providing pre-calculated skill affordances to the LLM, we would supply only atomic action affordances and the definition of skills. The LLM would then be responsible for reasoning, determining skill affordances, and allocating skills to the collaborators.

We conducted this experiment using GPT-4o and analyzed the **Correctly Allocated Skills Rate (CASR)**—the ratio of correctly allocated skills to the total number of skills per floorplan. We used the output of Component 3 of the framework (the generated Python script) from our original study and asked GPT-4o to allocate skills to collaborators. As previously mentioned, 40 experiments were conducted per GPT model. For GPT-4o, a total of 282 skills were involved across these 40 experiments, distributed as follows: 187 in the Kitchen, 34 in the Living Room, 16 in the Bedroom, and 45 in the Bathroom floorplans. The results were compared to those from the rule-based task allocation in the original study, as well as a random allocation baseline where skills were assigned arbitrarily between collaborators.

CASR results per floorplan and the overall CASR results are shown in Table VI. Out of 282 skills, 215 were correctly assigned to collaborators by the LLM, resulting in an overall CASR of 0.76. This indicates that while GPT-4o

	K-CASR	L-CASR	Bd-CASR	Bt-CASR	Overall-CASR
GPT-4o	0.71	1.00	1.00	0.71	0.76
Random	0.49	0.58	0.50	0.51	0.51
Rule-based	1.00	1.00	1.00	1.00	1.00

TABLE VI: CASR Comparisons for LLM-based, Random-based, and Rule-based Task Allocation

	K-OER	L-OER	Bd-OER	Bt-OER	Overall-OER
GPT-4o	0.40	1.00	1.00	0.70	0.62
Random	0.00	0.00	0.20	0.00	0.02
Rule-based	1.00	1.00	1.00	1.00	1.00

TABLE VII: OER Comparisons for LLM-based, Random-based, and Rule-based Task Allocation

effectively handles Task Planning (Component 2) and Code Generation (Component 3), its performance in reasoning over affordances for task and skill allocation remains mediocre. In this aspect, the rule-based method from the original study outperforms the LLM-based approach.

We are interested in studying the effect of incorrect skill allocations on the overall performance of the framework. When a skill is incorrectly allocated to a collaborator, two possible scenarios arise. In the first case, the collaborator possesses the skill but executes it at a higher cost compared to another collaborator. In the second case, the collaborator does not possess the skill and is unable to execute it. The first case results in a successful but suboptimal Task Execution—while the framework achieves the intended goal, it does so with increased cost. The second case, however, leads to immediate failure, as the task cannot be completed. Regardless of whether Task Execution is suboptimal or fails outright, we are primarily interested in measuring the rate of optimal Task Execution. To quantify this, we introduce a new metric: the **Optimal Execution Rate (OER)**.

Incorrect skill allocations by GPT-4o led to 15 failed or suboptimal executions out of 40 experiments during execution. This means that 25 out of 40 instructions were executed optimally (with an OER of 0.62). In contrast, the original study had 40 optimal executions, achieving a 1.0 OER.

VI. CONCLUSION

In this paper, we introduced a novel, adaptive framework that combines LLMs with continuous human feedback to dynamically refine task plans for human-robot collaboration (HRC). In contrast to contemporary methods, our work addresses challenging problems where humans provide natural language task specifications that are vague and for which the task objectives are implicit rather than explicit. We demonstrated that a single, concise prompt can effectively address diverse tasks and environments despite these challenges, eliminating the need for lengthy, structured prompts. By integrating human feedback directly into the planning process in an online fashion, our method is able to generate plans that are both efficient and aligned with human intentions. We evaluated our framework through various experiments conducted in the AI2-THOR simulation environment.

REFERENCES

[1] K. Erol, D. Nau, and J. Hendler, "Toward a general framework for hierarchical task-network planning," in *AAAI Spring Symp.*, 1993.

[2] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *Proc. Nat'l Conf. on Artificial Intelligence (AAAI)*, 1994, pp. 1123–1128.

[3] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: An HTN planning system," in *J. of Artificial Intelligence Research*, vol. 20, 2003, pp. 379–404.

[4] B. Nooraei, C. Rich, and C. L. Sidner, "A real-time architecture for embodied conversational agents: beyond turn-taking," in *Proc. Int'l Conf. on Advances in Computer–Human Interactions (ACHI)*, 2014, pp. 381–388.

[5] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proc. ACM/IEEE Int'l. Conf. on Human-Robot Interaction (HRI)*, 2015, pp. 205–212.

[6] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[7] V. Unhelkar and J. Shah, "Semi-supervised learning of decision-making models for human-robot collaboration," in *Proc. Conf. on Robot Learning (CoRL)*, 2020.

[8] S. Huang, X. Liu, Q. Zhang, and C. Wu, "Probabilistic decision model for adaptive task planning in human-robot collaborative assembly based on designer and operator intents," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 2236–2251, 2021.

[9] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Proceedings of the Conference on Robot Learning (CoRL)*, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78, 2017, pp. 262–270.

[10] b. ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do as I can, not as I say: Grounding language in robotic affordances," in *Proc. Conf. on Robot Learning (CoRL)*, 2023, pp. 287–318.

[11] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "PALM: Scaling language modeling with pathways," *J. Machine Learning Research*, vol. 24, no. 240, 2023.

[12] S. S. Kannan, V. L. N. Venkatesh, and B.-C. Min, "SMART-LLM: Smart multi-agent robot task planning using large language models," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[13] C. Breazeal, "Toward sociable robots," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 167–175, 2003.

[14] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu, "Robovie: An interactive humanoid robot," *Industrial Robot: An International Journal*, vol. 28, 12 2001.

[15] H. Ishiguro, T. Ono, M. Imai, and T. Kanda, "Development of an interactive humanoid robot "Robovie" — an interdisciplinary approach," in *Robotics Research*, 2003, pp. 179–191.

[16] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, 1999, pp. 968–973.

[17] Z. Xiao, A. Herzig, L. Perrussel, H. Wan, and X. Su, "Hierarchical task network planning with task insertion and state constraints," in *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, 2017, pp. 4463–4469.

[18] N. Gopalan and S. Tellex, "Modeling and solving human-robot collaborative tasks using pomdps," in *RSS Workshop on Model Learning for Human-Robot Communication*, 2015, pp. 590–628.

[19] H. Kurniawati, "Partially observable markov decision processes and robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 253–277, 2022.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int'l Conf. on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] B. Woodworth, F. Ferrari, T. E. Zosa, and L. D. Riek, "Preference learning in assistive robotics: Observational repeated inverse reinforcement learning," in *Machine learning for Healthcare Conference*, 2018, pp. 420–439.

[23] Y. Jang, J. Lee, and K.-E. Kim, "GPT-Critic: Offline reinforcement learning for end-to-end task-oriented dialogue systems," in *Proc. Int'l Conf. on Learning Representations (ICLR)*, 2022.

[24] G. Ye, H. Zhao, Z. Zhang, X. Zha, and Z. Jiang, "LSTDial: Enhancing dialogue generation via long-and short-term measurement feedback," in *Proc. Conf. of the North American Chapter of the Assoc. for Computational Linguistics (NAACL)*, 2024, pp. 5857–5871.

[25] J. Fang, X. Tan, S. Lin, I. Vasiljevic, V. Guizilini, H. Mei, R. Ambrus, G. Shakhnarovich, and M. R. Walter, "Transcrib3D: 3D referring expression resolution through large language models," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[26] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *Proc. Int'l Conf. on Learning Representations (ICLR)*, 2023.

[27] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An interactive 3D environment for visual AI," *arXiv preprint arXiv:1712.05474*, 2017.

[28] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, "LLM is like a box of chocolates: The non-determinism of ChatGPT in code generation," *arXiv preprint arXiv:2308.02828*, 2023.